

Qoopershare

Qoopershare.com is a website consisting of a distributed database of arbitrary data records, organized in master-detail trees (tables) of arbitrary depth. Each record contains fields: text, numeric, boolean, image. Each table is optionally associated with a screen layout file coded using a language called Coopertags: a simplified version of HTML. Every Qoopershare user can access databases belonging to other users; the databases live in shared folders which make use of Google Drive/Dropbox. For a non-technical description of Qoopershare, please click on Summary.

Business Model

Subscribers pay \$10/year or \$3.60 for 3 months. They can download records belonging to other users, and can copy/paste text contained in those records. They also have access to normal versions of image files contained in those records. Non-subscribers can only access thumbnail versions of image files, which have a maximum size of 9216 pixels (96 x 96). They cannot download records or copy/paste text. All users who join in the first 6 months, post-launch, receive a free 6-month subscription.

Cooperscript

Every database is associated with a Cooperscript web app which takes care of data entry and browsing the database. Cooperscript is a new open source web programming language employed by Qoopershare users. Database owners choose an existing Cooperscript app (and may customize it), or they can develop a Cooperscript app from scratch. To run Qoopershare, the user launches the embedded Jetty web server, and then uses a web browser, pointing it to localhost.

Searching

To search a distributed database for matching records, the user fills out one or more fields in one or more data entry screens, one screen per table. All radio buttons are initially deselected, and check boxes are 3-state: on, off, don't care. To match a null string in a text field, enter a single double quote character ("). Only alphanumeric characters are recognized in text edit boxes, other characters are treated as blanks. Keywords entered result in a hit whenever all keywords occur in the matching record. Phrases (multiple adjacent keywords) are delimited with double quotes (").

Search Results

The search results are returned in user-id/hit-count pairs. This list is sorted in descending order by hit-count. The hit-count can be view-count, zip-count, or view-zip-product-count. Whenever an end-user clicks on Zip, image data or some other piece of data is downloaded from another user's shared drive to the local drive, and the zip-count Z is incremented. Whenever an end-user performs a command resulting in a Zip button being displayed, the view-count V is incremented. The view-zip-product-count, $P = V(Z + 1)$.

Clicking on a list member in the list of user-id/hit-count pairs displays the first record (text, image, or both) matching the query. Clicking on next/previous steps through the list of records matching the query, which is sorted in descending order by hit-count. Text can be either plain text or Coopertags code (which may contain embedded images).

Indexed Search

Keywords

- keyid (4)
- text (16)

KeyPages

- keyid (4)
- pgid (8)

Phrases

- pgid (8)
- keyid (4)
- pos (2)

AppPages

- appid (4)
- pgid (8)

Monospace Mode

Monospace and rich-text modes pertain to local single-user Qoopershare scripts (accessed via localhost) created using Cooperscript and Coopertags. In monospace mode, all body text rendered to the screens of end-users is in a mono-spaced, typewriter-style font. Every character takes up 2 square cells: an upper cell and a lower cell. Superscripts and subscripts are handled by employing a vertical offset of one square cell. Header text is also mono-spaced, and each character takes up 2 oversized square cells.

Additional Formatting

The grid of characters can be subdivided into panels, which can themselves be subdivided into more panels, and so on. Any panel can contain zero or more text boxes, which may overlap each other. Vertical grid lines each take up one square cell per row of square cells. Horizontal grid lines are displayed in the same pixel row as underscore characters. Any row of square cells containing a horizontal grid line which is 2 pixels wide is taller by exactly one pixel. The following bracket characters: () [] { } can be oriented vertically or horizontally, taking up a single column or row of at least 2 square cells, respectively. Widgets such as check boxes, radio buttons, and combo box arrows take up 4 square cells (2 by 2). Images, animations, and diagrams are contained in canvas objects, which can appear anywhere panels can appear.

Rich-Text Mode

In rich-text mode, a given header or paragraph of body text can consist of a single variable-width font. Paragraphs have before/after spacing, left/right indent, and line spacing (single, double, 1.5, etc.). Panels have margins on all 4 sides. In both rich-text and monospace modes, text is either rendered to the HTML5 canvas object, or converted on the fly from Coopertags to HTML. The conversion takes place for subscribers, whereas rendering text to the canvas object is used for non-subscribers, to prevent them from selecting text and then copy/pasting it. Some features like form fields and submit buttons use hidden HTML.

Qoopershare for Dummies

Qoopershare (pronounced cooper-share) lets you create apps written in Cooperscript. Other Qoopershare users can download those apps and run them. Since Cooperscript is a web programming language, the user must first launch the web server which is bundled with Qoopershare, and then type localhost into the address bar of a web browser. Users can query a central database, which displays a list of matching users sorted by popularity. Clicking on a user in the list displays a matching page (text, image, or both), then clicking next/previous steps through the page list, also sorted by popularity. Every user who wants to share data with other users must have a shared drive (using Google Drive/Dropbox). Qoopershare makes money from subscription fees. Subscribers can download all data from other users. Non-subscribers can only download thumbnail versions of images from other users.

Public Fields

Public fields are Cooperscript fields (method variables) which are declared in a var block, whereas private (inner) fields are declared in an ivar block. Public fields which include getters and/or setters cannot be modified directly using an assignment statement, except within the class in which they are declared. For a field called myfield, the corresponding getter method is called get-myfield, and the corresponding setter method is called set-myfield. For a boolean field called myfield, the corresponding getter method is called is-myfield. For a boolean field called is-myfield, the corresponding getter method is called get-myfield.

About Me

I am Mike Hahn, the founder of Qoopershare.com. I was previously employed at [Brooklyn Computer Systems](#) as a Delphi Programmer and a Technical Writer (I worked there between 1996 and 2013). At the end of 2014 I quit my job as a volunteer tutor at [Fred Victor](#) on Tuesday afternoons, where for 5 years I taught math, computers, and literacy. I'm now a volunteer math/computer tutor at [West Neighbourhood House](#). My hobbies are reading quora.com questions/answers and the news at cbc.ca. About twice a year I get together with my sister Cathy who lives in Victoria. She comes here or I go out there usually in the summer. A few months prior to starting my Qoopershare project I used to lie on the couch a lot, not being very active. Now I'm busy most of the time. I visit my brother Dave once a month or so and I also visit my friends Main and Steph once or twice a month. For 26 years I was depressed on and off (I'm a rapid cycler), but it largely vanished after I started my Qoopershare project.

Contact Info

Mike Hahn
Founder, Qoopershare.com
515-2495 Dundas St. West
Toronto, ON M6P 1X4

Country: Canada
Phone: 416-533-4417
Email: hahnbytes (AT) gmail (DOT) com
Web: www.hahnbytes.com

Implementation Steps

1. Implement Cooperscript 0.1, console-based
 - Token parsing and building program tree has already been implemented
2. Finish Cooperscript 1.0, console-based
3. Make Cooperscript web-based
4. Begin code editor development
5. Read Murach's Java Servlets and JSP book
6. Implement Jabblor: web-based Scrabble game, user vs. robot
 - Jabblor is currently console-based Java Scrabble game
7. Write Coopertags design specs
8. Recruit GitHub open source coders/testers
9. Implement CPTG-to-HTML converter
10. Implement monospace mode
11. Implement rich-text mode
12. Integrate Cooperscript with Coopertags (monospace/rich-text modes)
13. Implement COOP-to-JS converter
14. Implement web-based functionality
15. Design core Qoopershare website (closed source)
 1. Distributed databases
 2. Indexed search
16. Launch core website
17. Advertise using Google AdWords
18. Beta test Qoopershare
19. Implement Cooperscript SDK
 1. WYSIWYG Coopertags editor
 2. Cooperscript code editor
 3. Convert Jabblor from Java to Cooperscript
20. Design full Qoopershare website
21. Launch full website
22. Accept credit card payments

Picoopery

Picoopery is a sample Cooperscript app which maintains a database of images of people. Many images include a specific person, having a first and last name.

Metadata

Each user has a corresponding SQL database stored in the root image folder. Every record in that database contains the following fields:

- id
- nextid
- parid
- childid
- width
- height
- status
- filename
- firstname
- lastname
- description

Status of a folder is either normal, hidden, or restricted. Hidden folders and all sub-folders are not shared. Restricted folders and all sub-folders are NSFW (not displayed by default).

Qunit Folders

A qunit is a queue of image files contained in a folder, and qunit folders can contain other qunit folders recursively. Newly added image files go to the head of the queue in each qunit. Every qunit contains a special queue which is a subset of the main queue of images. Both the main and the special queues support image reordering commands: head, tail, move left, move right. The head of the queue is the leftmost image in the queue.

Commands

- **Enter** - down a level
- **Up Arrow** - up a level
- **Left/Right Arrow** - previous/next
 - qunit/screen/image/slide
- **Down Arrow** - toggle main/special
- **Ctrl+Down Arrow** - toggle slide mode
- **1** - first
- **0** - last
- **Shift+Up Arrow** - move to top
- **Shift+Left Arrow** - move left
- **Shift+Right Arrow** - move right
- **Shift+Down Arrow** - move to bottom
- **L** - like image: make it special
- **U** - undo L command
- **I** - insert image/qunit
- **D** - delete (slide mode)
- **Ctrl+D** - delete image/qunit
- **Ctrl+Z** - undo
- **Ctrl+X** - cut qunit
- **Ctrl+C** - copy qunit
- **Ctrl+V** - paste qunit
- **Ctrl+N** - new qunit
- **Ctrl+R** - rename qunit
- **J** - justify (left, center, right)
- **B** - bookmark qunit/access bookmarks
- **H** - hidden qunit on/off
- **X** - restricted qunit on/off
- **Y** - sync qunit
- **R** - go to random screen
- **S** - search
- **Q** - quit
- **F1** - cycle: menu/help/normal
- **F11** - fill screen
- **[X]** - close menu bar

Qunits Mode

Displays parent qunit name followed by an indented list of qunit names. Current qunit is highlighted (enclosed in square brackets). Qunit properties: name, description, img-flag. If img-flag is false, qunit contains no images, only other qunits. By convention, images stored in non-image qunits reside in a sub-qunit called "\$".

Tiles Mode

Whenever the user is in Qunits Mode and presses Enter when an image qunit is highlighted, the current mode becomes Tiles Mode. The display is divided into 3 rows of equal height (or n rows where $n > 1$). Each row contains images. All portrait-mode images are of equal height but of varying widths. Every landscape-mode image is the same width as the height of the row which contains that image. All images are separated by a white, one-pixel gap (user may increase pixel count of gap, globally). Clicking on an image will display it in Image Mode. Pressing Up Arrow makes the current mode become Qunits Mode.

Image Mode

Single image is displayed, expanded by the maximum amount available on the user's display. Pressing L or U modifies special flag if needed and the current mode becomes Tiles Mode. Pressing Up Arrow makes the current mode become Tiles Mode. Pressing Enter enables editing of caption, and/or toggling display of image captions, and/or toggling the filtering out of images which lack captions. The image-view count of the user who is the image owner is incremented, if different from the user viewing the image.

Slide Mode

Press Ctrl+Down Arrow to toggle between Tiles Mode and Slide Mode. Slide Mode displays between 1 and 3 images per slide. Each slide tries to fill the entire display. Click on an image in slide mode to enter image mode, then press D to delete the image from the slide. In slide mode, press I to insert an image. The next time the I command is used in image mode, the image is inserted into the slide. Various image arrangements (side-by-side, stacked, or some combination) are used automatically, depending on the aspect ratios of the images on the slide. New slides can only be inserted at the head of the slide list. One of the panels on a given slide (which contains 1 to 3 panels) can contain plain text or Coopertags code instead of an image.

Bookmark/Insert Commands

The Insert command displays the select-user web page. After selecting a user, the Local mode switch is off. Further Insert commands insert images/qunits into the current local qunit, and allow the user to change the current local qunit. The Quit command sets the Local mode switch back to on. The Bookmark command bookmarks the current qunit when Local mode is off (enabling bookmark parent list selection), and accesses the bookmark tree when Local mode is on.

Cooperscript

Cooperscript is a Python dialect in which all operators precede their operands, and parentheses are used for all grouping (except string literals, which are delimited with double quotes). Cooperscript source files have a .COOP extension. Coopertags files have a .CPTG extension. COOPERScript stands for Compact Object-Oriented Program Editor and Runtime System. Cooperscript is open source and implemented using Java.

Special Characters

- () grouping
- - used in identifiers
- ; end of stmt.
- : dot operator
- " string delimiter
- \ escape char.
- # comment
- Extra:
 - _ used in identifiers
 - \$ string prefix char.
 - { } block comment
 - {{ }} Coopertags comment

Version 0.1

- No inheritance
- No interfaces
- No IDE

Keyboard Aid

This optional feature enables hyphens, open parentheses, and close parentheses to be entered by typing semicolons, commas, and periods, respectively. When enabled, keyboard aid can be temporarily suppressed by using the Ctrl key in conjunction with typing semicolons, commas, and periods (no character substitution takes place). By convention, hyphens are used to separate words in multi-word identifiers, but semicolons are easier to type than hyphens. Similarly, commas and periods are easier to type than parentheses. Typing semicolon converts previous hyphen to a semicolon, and previous semicolon to a hyphen (use the Ctrl key to override this behaviour). Typing semicolon after close parenthesis simply inserts semicolon. The close delim switch automatically inserts a closing parenthesis/double quote when the open delimiter is inserted.

Coopertags

Coopertags is a simplified markup language used to replace HTML (also used to define screen layouts). Arbitrary Coopertags code can be embedded in the Cooperscript echo statement. Coopertags syntax, where asterisk (*) means repetition, is defined as follows:

- | | | |
|--------------------------|-----------------------|---------------------------|
| • Tags: | • Body: | • Call Cooperscript code: |
| ◦ [tag] | ◦ text | ◦ [expr: <expr>] |
| ◦ [tag: body] | ◦ [: text]* | ◦ [exec: <stmt>...] |
| ◦ [tag (fld val)*: body] | ◦ [(fld val)*: text]* | ◦ [coop: <path>] |

Differences from Python

- Parentheses, not whitespace
- Integration with Coopertags
- Operators come before their operands
- Information hiding (public/private)
- Single, not multiple inheritance
- Adds interfaces ("scool" defs.)
- Drops iterators and generators
- Adds lambdas
- Adds quote and list-compile functions, treating code as data
- Adds cons, car and cdr functionality

Grammar Notation

- Non-terminal symbol: <symbol>
- Optional text in brackets: [text]
- Repeats zero or more times: [text]...
- Repeats one or more times: <symbol>...
- Pipe separates alternatives: opt1 | opt2
- Comments in *italics*

Cooperscript Grammar

White space occurs between tokens (parentheses and semicolons need no adjacent white space):

<source file>:

- do ([<imp>]... [<def glb>] [<def>]... [<class>]...)

<imp>:

<import stmt> ;

<import stmt>:

import <module>...
from <rel module> import <mod list>
from <rel module> import all

<module>:

<name>
(: <name><name>...)
(as <name><name>)
(as (: <name><name>...) <name>)

<mod list>:

<id as>...

<id as>:

<mod id>
(as <mod id><name>)

<mod id>:

<mod name>
<class name>
<func name>
<var name>

<rel module>:

(: [<num>] [<name>]...)
<name> // ?

<class>:

- <cls typ><name> [<base class>] [<does>] [<vars>] [<ivars>] do (<def>...) ;
- abclass <name> [<base class>] [<does>] [<vars>] [<ivars>] do (<anydef>...) ;
- <scool><name> [<does>] [<const list>] do ([<abdef>]... [<defimp>]...) ;
- enum <name><elist> ;
- ienum <name><elist> ;

<cls typ>:

class
iclass

<does>:

(does <scool name>...)

<scool name>:

<base class>:
<name>
(: <name><name>...)

<const list>:

(const <const pair>...)

<const pair>:

(<name><const expr>)

<scool>:

scool
iscool

<def glb>:

gdefun [<vars>] [<ivars>] do <block> ;

<def>:

- <defun> (<name> [<parms>]) [<vars>] [<dec>] do <block> ;

<defimp>:

- defimp (<name> [<parms>]) [<vars>] [<dec>] do <block> ;

<abdef>:

abdefun (<name> [<parms>]) [<dec>] ;

<defun>:

defun
idefun

<anydef>:

<def>
<abdef>

<vars>:

(var [<id>]...)

<ivars>:

(ivar [<id>]...)

<parms>:

[<id>]... [<parm>]... [(* <id>)] [(** <id>)]

<parm>:

(<set op><id><const expr>)

<dec>:

(decor <dec expr>...)

<block>:

([<stmt-semi>]...)

```

<stmt-semi>:
  <stmt> ;

<jump stmt>:
  <continue stmt>
  <break stmt>
  <return stmt>
  return <expr>
  <raise stmt>

<raise stmt>:
  raise [<expr> [ from <expr> ] ]

<stmt>:
  <if stmt>
  <while stmt>
  <for stmt>
  <try stmt>
  <asst stmt>
  <del stmt>
  <jump stmt>
  <call stmt>
  <print stmt>

<call expr>:
  • ( <name> [<arg list> ] )
  • ( : <obj expr> [<colon expr>]...
    ( <method name> [<arg list> ] ) )
  • ( call <expr> [<arg list> ] )

<call stmt>:
  • <name> [<arg list> ]
  • : <obj expr> [<colon expr>]...
    ( <method name> [<arg list> ] )
  • call <expr> [<arg list> ]

<colon expr>:
  <name>
  ( <name> [<arg list> ] )

<arg list>:
  [<expr>]... [ ( <set op><id><expr> ) ]...

<dec expr>:
  <name>
  ( <name><id>... )
  ( : <name><id>... )
  ( : <name>... ( <id>... ) )

<dot op>: // 'dot', ':', both OK
  dot | :

<del stmt>:
  del <expr>

<asst stmt>:
  <asst op><target expr><expr>
  <set op> ( tuple <target expr>... ) <expr>

<asst op>:
  set | addset | minusset | mpyset | divset |
  idivset | modset |
  shlset | shrset | shruset |
  andbset | xorbset | orbset |
  andset | xorset | orset |
  = | += | -= | *= | /= |
  //= | %= |
  <<= | >>= | >>>= |
  &= | ^= | |= |
  &&= | ^= | ||=

<set op>:
  set | =

<target expr>:
  <name>
  ( : <name> [<colon expr>]... <name> )
  ( slice <arr><expr> [<expr> ] )
  ( slice <arr><expr> all )
  ( <crop><cons expr> )

<arr>: // string or array/lyst
  <name>
  <expr>

<obj expr>:
  <name>
  <call expr>

<if stmt>:
  • if <expr> do <block> [ elif <expr> do <block>]... [
    else do <block> ]

<while stmt>:
  while <expr> do <block>
  while do <block> until <expr>

<for stmt>:
  for <name> in <expr> do <block>

<try stmt>:
  • try do <block> <except clause>... [ else do
    <block> ] [ eotry do <block> ]
  • try do <block> eotry do <block>

<except clause>:
  except <name> [ as <name> ] do <block>

<return stmt>:
  return

<break stmt>:
  break

```

<continue stmt>:
continue

<paren stmt>:
(<stmt>)

<qblock>:
(quote [<paren stmt>]...)

<expr>:
<keyword const>
<literal>
<name>
(<unary op><expr>)
(<bin op><expr><expr>)
(<multi op><expr><expr>...)
(<quest><expr><expr><expr>)
<lambda>
(quote <expr>...)
<renum expr>
<cons expr>
<tuple expr>
<lyst expr>
<dict expr>
<bitarray expr>
<string expr>
<bytezero expr>
<bytes expr>
<target expr>
<obj expr>
<cast>

<quest>:
quest | ?

<unary op>:
minus | notbitz | not |
- | ~ | !

<bin op>:
<arith op>
<comparison op>
<shift op>
<bitwise op>
<boolean op>

<arith op>:
div | idiv | mod | mpy | add | minus |
/ | // | % | * | + | -

<comparison op>:
ge | le | gt | lt | eq | ne | is | in |
>= | <= | > | < | == | !=

<shift op>:
shl | shr | shru |
<< | >> | >>>

*Note: some operators delimited with
single quotes for clarity
(quotes omitted in source code)*

<bitwise op>:
andbitz | xorbitz | orbitz |
& | ^ | '|

<boolean op>:
and | xor | or |
&& | ^^ | '||'

<multi op>:
mpy | add | strdo | strcat |
and | xor | andbitz | xorbitz |
or | orbitz |
* | + | % | + |
&& | ^^ | & | ^ |
'||' | '|'

<const expr>:
<literal>
<keyword const>

<literal>:
<num lit>
<str lit>
<bytes lit>

<cons expr>:
(cons <expr> [<expr>])
(<crop><cons expr>)

<tuple expr>:
(tuple <expr>...)

<lyst expr>:
(lyst [<expr>]...)

<dict expr>:
(dict [<pair>]...)

<bitarray expr>:
(bitarray <enum name> [<elist>])
(bitarray <enum name><idpair>...)

<elist>:
<id>...
<intpair>...
<chpair>...

```

<intpair>
  // integer constant
  <int const>
  ( <int const><int const> )

<chpair>
  // one-char. string
  <char lit>
  ( <char lit><char lit> )

<idpair>
  <idt>
  ( <id><id> )

<pair>:
  // expr1 is a string
  ( <expr1><expr2> )
  ( <str lit><expr> )

<renum expr>
  ( renumize <expr><ren id>... )
  ( renumize <expr><ren int>... )
  ( renumize <expr><ren ch>... )

<ren id>:
  ( 0 <id> )
  ( 1 <id> )
  ( 1 <id><id> )

<ren int>:
<ren ch>:
  // expr is <dec int> | <char lit>
  ( 0 <expr> )
  ( 0 <expr><expr> )
  ( 1 <expr> )
  ( 1 <expr><expr> )

<cast>:
  ( cast <type><expr> )

<print stmt>: // built-in func
  print <expr>...
  println [<expr>]...
  echo <expr>...

<lambda>:
  ( lambda ( [<id>]... ) <expr> )
  ( lambda ( [<id>]... ) do <block> )
  ( lambdaq ( [<id>]... ) do <qblock> )
  // must pass qblock thru compile func

```

No white space allowed between tokens, for rest of Cooperscript Grammar

```

<white space>:
  <white token>...

<white token>:
  <white char>
  <line-comment>
  <blk-comment>

<line-comment>:
  # [<char>]... <new-line>

<blk-comment>:
  { [<char>]... }

<white char>:
  <space> | <tab> | <new-line>

<name>:
  • [<underscore>]... <letter> [<alnum>]...
    [<hyphen-alnum>]... [<underscore>]...

<hyphen-alnum>:
  <hyphen><alnum>...

<alnum>:
  <letter>
  <digit>

In plain English, names begin and end with zero or more underscores. In between is a letter followed by zero or more alphanumeric characters. Names may also contain hyphens, where each hyphen is preceded and succeeded by an alphanumeric character.

<num lit>:
  <dec int>
  <long int>
  <oct int>
  <hex int>
  <bin int>
  <float>

<dec int>:
  [<hyphen>] 0
  [<hyphen>] <any digit except 0> [<digit>]...

<long int>:
  <dec int> L

```

<float>:
 <dec int><fraction> [<exponent>]
 <dec int><exponent>

<fraction>:
 <dot> [<digit>]...

<exponent>:
 <e> [<sign>] <digit>...

<e>:
 e | E

<sign>:
 + | -

<keyword const>:
 none
 true
 false

<oct int>:
 0o <octal digit>...

<hex int>:
 0x <hex digit>...
 0X <hex digit>...

<bin int>:
 0b <zero or one>...
 0B <zero or one>...

<octal digit>:
 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7

<hex digit>:
 <digit>
 A | B | C | D | E | F
 a | b | c | d | e | f

<str lit>:
 [\$ <str prefix>] <quoted str>

<str prefix>:
 r | u | R | U

<quoted str>:
 " [<str item>]... "

<str item>:
 <str char>
 <escaped str char>
 <str newline>

<str char>:
 any source char. except "\", newline, or
 end quote

<str newline>:
 \ <newline> [<white space>] "

<bytes lit>:
 \$ <byte prefix><quoted bytes>

<byte prefix>: // any case/order
 b | br

<quoted bytes>:
 " [<bytes item>]... "

<bytes item>:
 <bytes char>
 <escaped char>
 <str newline>

<bytes char>:
 any ASCII char. except "\", newline, or
 end quote

<escaped char>:
 \\ *backslash*
 \" *double quote*
 \} *close brace*
 \a *bell*
 \b *backspace*
 \f *formfeed*
 \n *new line*
 \r *carriage return*
 \t *tab*
 \v *vertical tab*
 \ooo *octal value = ooo*
 \xhh *hex value = hh*

<escaped str char>:
 <escaped char>
 \N{name} *Unicode char. = name*
 \uxxxx *hex value (16-bit) = xxxx*

<crop>:
 c <crmid>... r

<crmid>:
 a | d