

```
--: {field: []: (😊) <- Field Functions Library: Produce and Execute Field Plans ->
muse/docs/lib/field.md
--😊 field: Fields are rectangular solids defined by a range (a situation pair with fields keyed
properties). -> field, _field
--:+ Fields are made up of plots, each plot at least small enough to deal with turtle inventory
limitations.

--:# Execution train runs from field.make to field.plot to field.plan to execute the plan

--: field.make(commands: fieldCommands, faced: ^:) -> Load field files; return their field.plot
calls -> report: ":" &:

--:> fieldCommands: For CLI -> :[fieldOpName: ":", ranger: ":", firstPlot: #:?,
lastPlot: #:??]
--:+ The second entry, ranger in fieldCommands is a string which may simply be a name for a
range, (a kind of place).
--:+ If so, the range name gets the range's features dictionary and the field file name to load (keyed as
features.fields).
--:+ It could also be a string specifying the name of a farm and a farm field name in that farm
(separated by a colon).
--:+ If so, the farm name specifies the farm's range and so the farm range's features dictionary.
--:+ The fields entry in that dictionary is itself a dictionary, keyed by the farm's field name to specify
its range name.
--:+ With the proper range name in hand, either directly as above, or from the farm, the field file to
load is specified.

--: field.plot(commands: field.plotSpan, fieldsOp: (😊, fieldOpName: ":" , plots: #:, offset: xyz?) ->
Plots -> report: ":" &: &!
--:+ Called by field files. Calls fieldsOp from field file (which calls field.plan).

--:> field.plotSpan: {} spans all plots; if only first, default plots after first -> :[_:, _:, first: #:?,
last: #:??]

--: field.plan(planName: ":" , fielding: fieldParameters, offset: xyz?) -> Run plan, default offset {0,0,0}.
-> report: ":" &: &!
--:+ Loads and executes the prototype plan (which calls field.paths) for each (odd, even, or last)
level of a plot.

--:> fieldParameters: bounds (and materials to fill and replace) -> :[bounds,
fieldParameters.fills?, fieldParameters.removeables??]

--:> fieldParameters.fills: Group or list of craft items for fill material -> group|craft[]

--:> fieldParameters.removeables: Material replaced by fill -> group|craft[]

--:> craft: Minecraft item detail.name without minecraft: prefix -> ":"

--: field.extents(:bounds:, :strides:, faced: "?:") -> Plots placed -> field.count, field.count ,
eP, eP, striding, ^:, ^:
```

--: Returns `nplots:[fieldOp #:]`, `slots:[fieldOp]: #:]`, `strides: eP`, `run: eP`, `striding`, `turn: ^:`, `back: ^:`

--: Extends for *stride* (shorter) and *run* (longer) virtual axes for each *opName* in the *strides* entries unless *faced*.

--: field.count: _dictionary keyed by 'opName' for number of elements in field for that operation_ -> [fieldOp]: #:`

--: strides: dictionary keyed by *opName* for the distance along the stride axis for a striding -> [fieldOp]: #:

--: striding: dictionary keyed by *opName* of vectors incrementing game coordinate positions for *turn* -> [fieldOp]: xyz

--: eP: pair of coordinates for extents -> :[xyz, xyz]

--: fieldOp: Operation name in the set for a particular kind of field -> ":"

--:# **Path generator: flying ox plow paths through given three dimensional rectangular bounds.**

--: Ox plow paths minimize travel to plow a field. Flying oxen (aka turtles) do that in three dimensions.

--: field.paths(bounds: xyz[]) -> Called by plan prototype file to generate plans for plot. -> paths, yDelta: #:, xzEdge: facing

--: paths: _Flying ox traverse of three dimensional rectangular solid -> {start: ":"[], odd: ":"[], even: ":"[], last: ":"[]}

--: Returns paths, vertical traverse (*yDelta: #:*), and orientation of longest horizontal edge for bounded block.

--:# **Cut, fill, till, and traverse points defining rectangular volumes** using `field.plan` to optimize traversal.

--: field.cut(places: [:nearPlace: ":", farPlace: ":"]) -> Quarry out blocks from one place to the other. -> ":" &:

--: cut point point -> Quarry out blocks bound by named points (defining a rectangular solid).

--: field.fill(parameters: [:nearPlace: ":", farPlace: ":", fill: ":", target: ":"?]) -> Fill, Till, Replace. -> ":" &:

--:< Filling and target may be one of the turtle categories or a Minecraft detail name without prefix `minecraft:`

--: fill point point filling ?target -> Layer fill bounds by points; optionally swaps out only target blocks.

--: field.till(parameters: [:nearPlace: ":", farPlace: ":", seed: ":"]) -> Till the seed from one place to the other. -> ":" &:

2025-11-25

--:< *Seed may be one of the turtle categories or a Minecraft detail name without the prefix*

"minecraft:"

--:- till point point seed -> *Till the seed bounds by named points (defining a rectangular solid).*

--:: field.fence(parameters: :[ranger: ":", fencing: ":@"?]) -> *Put fencing using **Layer** plan. -> " : "*

--:- fence range [item] -> *Put item or available wooden fence from one point to another in range.*