Bluebeam JavaScript Header Fields
Two-Line Input Limits & Page-to-Page Syncing

This document explains **why** Bluebeam header fields behave the way they do, and **how** we control that behavior using JavaScript.

---

1. Why Header Fields Are “Per Page”

In Bluebeam (and Acrobat), a form field can exist in two fundamentally different ways:

- **Single field with multiple widgets** (same name on every page)
- **Multiple independent fields** (unique name per page)

This script intentionally uses the second approach:

```
ProjectName.0
ProjectName.1
ProjectName.2
```

Why?

- Allows different values per sheet if ever needed
- Avoids unpredictable behavior when pages are reordered
- Works reliably with mixed page sizes (Letter, 11x17, etc.)

> **Important:**
> Because these are *different fields*, they do NOT automatically stay in sync. Any syncing must be done intentionally with JavaScript.

---

2. Limiting a Field to Two Lines (Important Gotcha)

The Problem

Bluebeam does **not** have a built-in "max lines" setting. If a field is multiline, the user can normally:

- Press Enter indefinitely
- Paste paragraphs of text
- Scroll text out of view

The Solution (Two Layers of Protection)

We enforce a two-line limit using:

1. **Keystroke action** – blocks typing or pasting beyond two lines
2. **Validate action** – trims excess lines if something slips through

Why Two Layers?

> Keystroke catches normal typing.
> Validate catches copy-paste, undo/redo, or edge cases.

The Code

```
// Prevent typing or pasting more than 2 lines
function PB_limitLinesKeystroke(maxLines) {
  var cur = String(event.value || "").replace(/\r\n/g, "\n");
  var chg = String(event.change || "").replace(/\r\n/g, "\n");

  var before = cur.substring(0, event.selStart);
  var after  = cur.substring(event.selEnd);
  var proposed = before + chg + after;

  if (proposed.split("\n").length > maxLines) {
    event.rc = false;
  }
}

// Final cleanup safeguard
function PB_enforceMaxLinesValidate(maxLines) {
  var v = String(event.value || "").replace(/\r\n/g, "\n");
  var lines = v.split("\n");
```

```
  if (lines.length > maxLines) {
    event.value = lines.slice(0, maxLines).join("\n");
  }
}
```

How It's Applied

```
f.multiline = true;
f.doNotScroll = true;
f.setAction("Keystroke", "PB_limitLinesKeystroke(2);");
f.setAction("Validate",  "PB_enforceMaxLinesValidate(2);");
```

Result:

- User can type exactly two lines
- Third line is blocked
- Header text never scrolls out of view

---

3. Syncing Page 1 to All Other Pages

The Goal

When the user edits:

```
ProjectName.0
```

We want:

```
ProjectName.1
ProjectName.2
ProjectName.3
```

to automatically match it.

Why We Use Page 0 as the "Master"

- Clear mental model: edit once
- No circular updates

- No race conditions

The Sync Function

```
function PB_syncHeaderField(baseName) {
  var master = this.getField(baseName + ".0");
  if (!master) return;

  for (var p = 1; p < this.numPages; p++) {
    var f = this.getField(baseName + "." + p);
    if (f) f.value = master.value;
  }
}
```

When Sync Happens

We attach it to the **Validate** event on page 0:

```
this.getField("ProjectName.0")
  .setAction("Validate", "PB_syncHeaderField('ProjectName');");
```

That means:

- User finishes typing
- Clicks away
- All pages update instantly

---

4. Why We Make Other Pages Read-Only

If every page stayed editable:

- Edits could fight each other
- Last edit wins (unpredictable)
- User confusion

Best Practice

- Page 1 (index 0): Editable
- All other pages: Mirrors only

```
function PB_setMirrorsReadonly(baseName) {
  for (var p = 1; p < this.numPages; p++) {
    var f = this.getField(baseName + "." + p);
    if (f) f.readonly = true;
  }
}
```

---

5. Why This Design Is Reliable

- Page reordering does not break headers
- Mixed page sizes work correctly
- Headers never scroll or overflow
- User edits exactly one location

> This approach mirrors how professional calculation packets, drawing headers, and spec covers are typically automated.

---

6. Common Questions

“Why not just use one field name everywhere?”

Because Bluebeam will treat them as one logical object. Page reordering and resizing can produce unpredictable results.

“Why not trust char limits alone?”

Character limits do not control line breaks. Two short lines can exceed one long line visually.

“Why Validate instead of Keystroke only?”

Paste, undo, and scripted changes bypass keystroke. Validate is the safety net.

---

7. Summary

- Two-line limits require JavaScript

- Per-page fields require explicit syncing
- Page 0 is the master
- Validate events are key

This setup is deliberate, robust, and designed for real-world calculation packets.