Revision one

This page and the next were written by an ordinary Save. Nothing about them knows anything is going to be added, and nothing about them has to: an incremental update leaves the bytes it was given exactly as they were and writes after them.

A reader opening the finished file starts at the last startxref, follows /Prev backwards through the cross-reference sections, and takes the first definition it finds of each object - so a later revision shadows an earlier one without erasing it. Everything on this page is still the first definition of itself; nothing later contradicts it.

What an incremental update is for

Three things, and file size is not one of them. A signed document can only be changed this way, because a signature covers a byte range of the file and rewriting the file invalidates it. An audited document keeps every earlier state recoverable. And a very large document can be annotated without being written out again from end to end.

What it costs

The file only ever grows, and it grows by more than the change: every object touched is written again in full, and so is a whole cross-reference section. A document revised a hundred times carries a hundred copies of whatever kept changing. Rewriting it with Save is how that is reclaimed, and is exactly what must not be done to a signed one.

Append is a mode, not a flag

PdfDocumentOpenMode.Append is the only mode a document can be appended to in, and the reason is object numbers. Modify reads everything into memory and renumbers it, so an appended definition of object 12 would shadow whatever happened to be numbered 12 this time round - which is not what it was numbered before. Import does not keep the bytes at all. Append keeps both, and SaveIncremental refuses without them.

Only what changed is written

Which means something has to know what changed. An object modified through the object model marks itself; a direct one - an array held inside a page dictionary rather than indirectly in its own right - cannot, because changing it changes the page and not the array. That is what PdfObject.MarkAsChanged is for, and forgetting it on the page is how an appended annotation ends up in a file no reader shows it in.

The trap worth knowing before you hit it

SaveIncremental writes the whole file - original bytes and all - so the destination has to be empty. Handing it the file it was read from is the tempting mistake and the damaging one: the original is rewritten over itself, the revision is appended, and because nothing truncates, whatever of the old file ran past the new end survives. That includes its startxref, which a reader scanning backwards finds first - and the appended revision is then ignored in silence, signature and all. So it throws on a non-empty stream rather than letting that happen.

Revision two

Added by appending, not by rewriting

This page did not exist when the bytes above it were written. The document was opened again with PdfDocumentOpenMode.Append, this page was added, the subject line in the information dictionary was changed, and SaveIncremental wrote the original bytes through untouched with a new cross-reference section after them.

Revision one	`33,325 bytes`
Objects reachable now	`29 across both revisions`
Pages before this one	`2`
Changed as well as added	`/Info /Subject, which revision one had left empty`

More changed than the page

Adding a page is not only a new page object. The /Pages node that lists them has to say so, and its /Count has to agree, so the appended revision carries a second definition of the page tree node as well. The larger cost is the fonts: a document opened for appending does not adopt the font objects already in the file, so a page drawn in a face the first revision also used embeds that face again. Compare the two sizes on the next page - the revision is far larger than anything visible on this one, and almost all of it is a second copy of the type.

Revision three

And this one was added to the file the last one produced. Below is what a byte scanner finds in those bytes - counted, not described, by looking through the file revision two wrote for the markers a reader uses to walk it.

Revision one	`33,325 bytes`
After revision two	`155,297 bytes`
Appended by revision two	`121,972 bytes`
startxref, at a line start	`2 - one per revision`
%%EOF, at a line start	`2 - one per revision`
/Prev, anywhere in the bytes	`1 - one fewer, and rightly so`
This file	`one revision deeper again`

Why /Prev is one short, and why two rows say where

Two revisions have two startxrefs and one /Prev, not two. Every revision writes a cross-reference section; every section but the first points at the one before it, and the first has nothing behind it to point at. So the chain has one fewer link than it has sections, and a file with a /Prev per revision would be one with a link into nothing.

The other caveat is how these were counted. A byte scan cannot tell a marker in a trailer from the same characters inside a compressed stream or an embedded font, and this file carries both - so the first two rows count the marker only where it begins a line, which is where a trailer puts it and where a stream almost never does. The third is a plain count and is worth exactly that much. Structure is what a reader parses for; this page is looking at the file rather than reading it.

Reading it back

Nothing special is needed. A reader that understands PDF at all understands an incrementally updated file, because following /Prev is how cross-reference sections have always been chained - a linearised file has more than one section too. Open the finished PDF in anything and it is a four page document; the three revisions are visible only to something looking at the bytes.

What an earlier revision still holds

Everything it ever said. Redacting a document by drawing a black rectangle over a name and appending the change leaves the name in the file, in plain text, one revision back - and tools that recover it are not sophisticated. Redaction means rewriting, which means Save, which means any signature goes with it. That tension is real and has no trick to it: the two features want opposite things.

Where this meets signing

PdfSigner does exactly what this page does - it appends a revision - and that is the whole reason signing a document that was already signed does not destroy the first signature. See the Signing demo, whose output is one revision written the same way with a hole patched into it.