bulk_extractor 2.2 Developer Manual

bulk_extractor project

July 2026

Scope

This manual describes the 2.x source tree, Autotools build, and scanner API. The repository's `README.md` and `doc/scanner_api.md` are the authoritative references and are updated with the source.

Build and test

For a Git checkout, install the dependencies listed in the README, then run:

```
bash bootstrap.sh
./configure
make
make check
```

Release archives already contain `configure`, so they do not require `bootstrap.sh`. Use `make distcheck` before a release or when changing the build/distribution manifests. The source tree includes the be20 API and DFXML dependencies; do not initialize recursive submodules.

Source layout

- `src/` contains the executable, built-in scanners, and the integrated `be20_api` component.
- `doc/scanner_api.md` defines the scanner contract and `doc/scanner_template.cpp` is the loadable-scanner starting point.
- `tests/` and the project Makefiles contain regression and integration tests. Run them through `make check`.
- `.github/workflows/` contains the supported CI builds, including the MinGW Windows artifact build.

Scanner lifecycle

A scanner is a `scanner_t` function that switches on `scanner_params::phase`. In `PHASE_INIT`, call `sp.check_version()`, name the scanner, and declare feature recorders, histograms, flags, and scanner options. In `PHASE_INIT2`, retrieve declared recorders. `PHASE_SCAN` may run concurrently on multiple worker threads, including concurrent calls to the same scanner. Treat `sp.sbuf` as

read-only and valid only during that callback. Finish scanner-owned output in PHASE_SHUTDOWN and release state in PHASE_CLEANUP; cleanup also runs for disabled scanners.

Declare feature recorders only in PHASE_INIT; retrieve them in PHASE_INIT2; write findings during PHASE_SCAN. Recorder writes are safe concurrently, but other shared scanner state needs appropriate locking or atomics. A scanner can be bypassed for ineligible buffers, so it must not assume it sees every input buffer.

Loadable scanners

Copy doc/scanner_template.cpp, rename its scanner and metadata, and build it against the same bulk_extractor source version as the executable. A module named scan_* exports:

```
extern "C" scanner_t *bulk_extractor_scanner_v1();
```

Use -P to load a development module from an explicit directory. The loader also searches BE_PATH. Keep all factory and lifecycle rules from doc/scanner_api.md; do not retain scanner_params or pointers into an sbuf after a callback returns.

Windows executable

The project does not support a native Windows build or an installer. The Windows MinGW build workflow cross-compiles a 64-bit executable on Ubuntu using the x86_64 MinGW-w64 POSIX toolchain. It builds static Expat, RE2, and Abseil through a pinned vcpkg checkout, configures with --host=x86_64-w64-mingw32 --disable-libewf, and links the executable with -all-static. The workflow uploads a Windows artifact containing bulk_extractor64.exe.

The CI job rejects unexpected non-system DLL imports. A Windows runner then downloads and executes that same artifact against an input directory with a Unicode filename. The artifact does not include libewf, so it cannot read E01 images. Treat it as a CI artifact, not a signed release package.