Win95 USB Lab

Reproducible VM Testing for Windows 95 OSR2

Technical white paper · 13 September 2026 · Revision 1.0

Prepared for Luke Kist. Describes software release **0.1.0-preview.1**.

Abstract

Win95 USB Lab provides a small, inspectable framework for evaluating USB input, USB storage, and optional network connectivity in an existing Windows 95 OSR2 virtual machine. A native guest executable records device status, application input, file integrity, and network exchanges. Modern host tools prepare uniquely identified test media, inspect executable imports, and evaluate the resulting reports. A three-phase procedure is designed to repeat the selected checks after a Windows restart and after a cold boot.

The central design principle is to require evidence appropriate to each claim. A device name does not establish a started driver. Application clicks do not identify their transport. Successful file access does not establish a USB attachment. A shutdown request does not prove that a machine stopped. The framework therefore combines guest diagnostics with a separate requirement for observed device routing and lifecycle evidence.

The preview has passed 20 host-side tests. Its release archive was rebuilt byte for byte from an extracted copy using the same build environment, its prepared media were read back successfully, and its executable's imported API names were found in locally preserved OSR2 DLL exports. These results establish a reviewed preparation and reporting baseline. The new preview has not yet been executed in a live guest. USB HID compatibility remains unresolved, storage I/O across boots is unverified, and physical hardware testing is deferred.

The intended contribution is reusable test infrastructure for the Windows 95 community. The package supplies original source and a diagnostic executable; compatible USB drivers and a licensed operating-system installation remain separate prerequisites.

Status as of 13 September 2026

Area	Evidence available
Preparation, checks, packaging	20 passing host-side tests; reproducible archive in the recorded environment
Guest API surface	Static import/export audit passed; runtime behavior unverified
Live VM USB storage and networking	Test procedure prepared; execution pending
USB mouse/tablet	Diagnostic prepared; compatible driver unresolved
Physical USB hardware	No tests performed; no compatibility claim
Distribution	Local preview archive prepared; GitHub publication pending

This is a technical description of an experimental preview, not a USB driver release or a hardware certification. No throughput, latency, or reliability benchmark is claimed.

1. Problem, scope, and reference platform

1.1 A claim must identify the complete path

For this project, a useful USB compatibility result connects a specific guest build, controller, peripheral, driver stack, and observed operation. An installation wizard accepting an INF is an early observation. A useful final result also shows the driver started, the intended operation completed, and the configuration survived the specified lifecycle transitions without renewed driver prompts.

Storage and input require separate treatment. The XUSBSUPP project by Conner McCoy and Rudolph Loew documents an OSR2 storage path incorporating RLUSB9X. Its package documentation explicitly excludes USB HID keyboard, mouse, and joystick drivers, as well as USB 2.0 and USB 3.0 controller support. Those are statements about that package; they are not a survey of every driver that may exist today. Win95 USB Lab obtains no HID capability merely by using the storage supplement. Source: XUSBSUPP authors' documentation.

The preview makes USB input, USB storage, and their combined execution selectable suites. PCnet networking is an additional opt-in check. In particular, the network test exercises an emulated PCI NIC; it does not test a USB Ethernet adapter.

1.2 Initial qualification target

Component	Reference configuration
Operating system	Existing licensed Windows 95 OSR2; version 4.0, build 1111 or later within the Win95 build gate
Machine	QEMU PC with ACPI disabled
Processor and memory	One Pentium II CPU; 64 MiB RAM
Display	Standard VGA; at least 640 × 480
USB controller	PIIX3 UHCI, USB 1.x
Input candidate	QEMU USB tablet; PS/2 recovery retained
Storage candidate	USB bulk-only storage backed by a dedicated FAT16 image
Optional network	AMD PCnet with QEMU user-mode NAT; no inbound forwarding

The profile is recorded in `profiles/qemu-osr2.json`. It describes the intended test configuration and does not launch a VM. QEMU documents its tablet as an absolute-coordinate pointer and its `usb-storage` device as bulk-only storage backed by a separate drive object. The proposed profile explicitly routes the candidate devices through UHCI. Source: QEMU USB emulation documentation.

Windows 95 RTM, other Windows releases, other hypervisors, EHCI/xHCI controllers, and physical peripherals fall outside this preview's qualification scope. The checker also recognizes particular PCI identities and driver names. A different platform or driver stack requires an explicit checker/profile revision and a new qualification run.

2. Architecture and distribution

2.1 Three cooperating components

The architecture separates preparation, guest observation, and VM ownership. Modern host tools handle files and report evaluation. The native executable performs operations inside Windows. The existing VM manager owns launch, device attachment, input routing, restart observation, shutdown, and cleanup.

The host entry points are `prepare.py` for fresh runs and media, `check.py` for guest reports, `audit.py` for API-name inspection, and `package.py` for an allowlisted release archive. The guest component is `U95TEST.EXE`.

The evidence flow is:

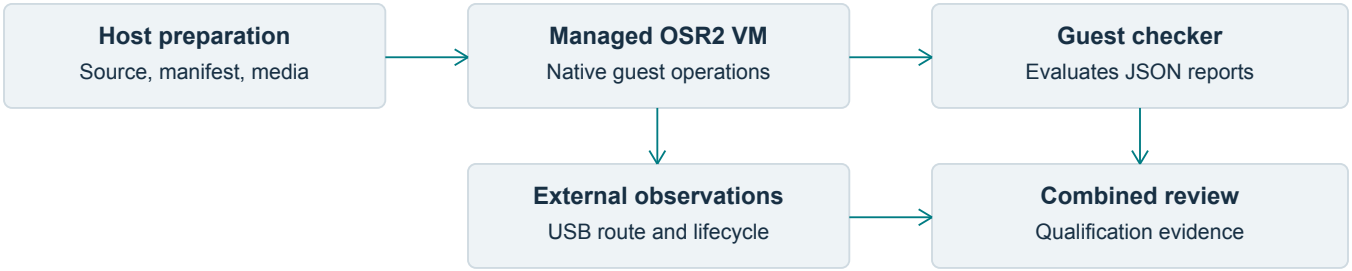

Figure 1. Guest diagnostics and external VM observations supply different parts of a qualification record. The portable checker does not merge them into an automatic live qualification.

The tools do not start QEMU, create a replacement operating-system disk, or open a raw host USB device. In the project's own environment, Perslis/GhostBridge retains lifecycle and observation ownership. The public kit can be used with an existing VM workflow without importing the Perslis-specific controller code.

2.2 A small legacy executable

The probe is built from C as a 32-bit x86 Windows executable with OS and subsystem version 4.0 headers. It calls Windows APIs directly and supplies its own small memory helpers, avoiding a C runtime dependency. Configuration Manager entry points are resolved dynamically from `CFGMGR32.DLL`; network operations use Winsock 1.1.

This design makes the imported surface inspectable. It does not make modern compiler output automatically compatible with every Win95 installation. The recorded API audit checks the candidate executable against DLLs preserved from the target installation; actual loading and execution remain a separate test.

2.3 What is distributed

The archive contains original C, Python, and shell source; documentation; a profile; host tests; and `U95TEST.EXE`. Original material is MIT licensed. The archive excludes Windows DLLs, third-party drivers, installation cabinets, operating-system images, saved memory, and private machine configuration. Dependencies and third-party licensing boundaries are documented separately in `THIRD_PARTY.md`.

3. Guest diagnostics and controlled file operations

3.1 Device state and ancestry

The probe walks the Configuration Manager device tree and records identifiers, parent relationships, status-return codes, problem codes, class, driver key, and selected driver binding values. Traversal has depth and node limits; an incomplete enumeration is not accepted as complete evidence.

The checker requires successful status retrieval, `DN_STARTED`, absence of `DN_HAS_PROBLEM`, a zero recorded problem value, and a nonempty driver key. It then checks the expected device identities and ancestry. Microsoft documents the status and problem outputs of `CM_Get_DevNode_Status`; its current documentation is useful for API semantics but does not establish historical Win95 availability. Source: Microsoft Configuration Manager documentation.

The reference path requires UHCI bound to `UHCD.SYS`, a descendant root hub bound to `USBHUB.SYS`, and the selected peripheral stack below that hub. Input expects `HIDUSB.SYS` with a `MOUHID` descendant; storage expects `RLUSB9X.SYS`. Optional networking expects PCnet and `PCNTN3.VXD`. These checks establish reported bindings and state, not an independent measurement of every loaded driver binary.

3.2 Input observed inside Windows

For input tests, the probe draws two separated targets and records Windows mouse-button down/up messages within defined coordinate tolerances. The executable does not synthesize cursor movement. A two-minute input timeout permits a failed attempt to produce a diagnostic report.

External input-route evidence is still essential: PS/2 input could reach the same window. A guest click becomes USB evidence only when the observer can connect that action to the intended USB tablet route. Keyboard Enter can advance a completed test and is never counted as a successful pointer action. The storage-only suite requires no pointer test.

3.3 Storage with a fresh run marker

Preparation creates a new auxiliary FAT16 image, approximately 32 MiB, with its partition beginning at sector 63. It contains a 32-character run identifier plus CRLF in `U95VOL.TAG`, and a deterministic 64 KiB `SEED.BIN`.

Before writing, the probe requires an exact marker match, a valid seed, and intact files from earlier phases. It then creates a new 64 KiB `U95P0.BIN`, `U95P1.BIN`, or `U95P2.BIN`, writes the deterministic phase pattern, flushes, closes, reopens, and verifies the bytes. Existing files are not overwritten. Host and C implementations of the pattern are checked against each other.

The selected drive must be D: through Z:. This restriction and the marker reduce accidental writes to the wrong volume; neither proves physical device identity. Successful file access must be paired with device-tree and attachment evidence showing that the image traveled through USB. The operations are bounded in data volume, but synchronous driver I/O can still hang.

4. Network checks and the three-phase protocol

4.1 Optional connectivity evidence

Networking is disabled by default. When enabled, the guest sends a UDP DNS query to the configured IPv4 server, then opens a TCP connection to the returned IPv4 address on port 80 and requests an HTTP response. Defaults are QEMU NAT DNS at `10.0.2.3` and the hostname `example.com`. The request includes the run identifier and phase, connecting the observation to the current test.

The checker requires the configured DNS exchange, a local address in the reference `10.0.2.0/24` network excluding designated reserved addresses, a globally routable remote IPv4 address, and a syntactically valid HTTP/1.0 or HTTP/1.1 status line. An HTTP error status can satisfy the transport check. This is neither a successful web-application transaction requirement nor an HTTPS test.

Socket waits and transfer loops use timeouts. They do not establish a universal deadline for every legacy API or driver call. Raw DHCP registry values are retained for inspection, but the checker does not decode them into proof of a DHCP lease. Lease and PCnet TCP/IP binding evidence remain external review requirements.

4.2 Installation and phase sequence

Installation validates the adjacent `TEST.INI`, creates `C:\U95TEST`, copies the probe and configuration, and adds its named Windows Run entry. It refuses a conflicting folder or startup entry. It installs no device driver. `/inspect` produces a device snapshot without consuming a phase, sending network requests, or writing storage test files.

Phase	Selected checks	Requested transition and required observation
0: Initial run	Device state and selected input/storage/network operations	Windows restart; observer verifies the guest restarts within the same VM process
1: After restart	Repeat operations; verify phase 0 storage data if selected	Windows shutdown; observer verifies safe shutdown, stops the emulator, and confirms process absence
2: After cold launch	Repeat operations; verify earlier storage files if selected	Remove test startup and request final shutdown; observer verifies cleanup

The guest writes `PHASE0.JSON` through `PHASE2.JSON` and corresponding `EXIT` request records. Its persistent phase counter advances when the executable runs. Consequently, three reports alone cannot prove three distinct boot conditions.

The transition uses `ExitWindowsEx`. Microsoft documents that successful return indicates initiation, not guaranteed completion. The protocol therefore requires observation beyond the request record. Current Microsoft documentation is cited for that distinction; it is not treated as a Win95 compatibility guarantee. Source: Microsoft ExitWindowsEx documentation.

A cold-boot result requires the guest to have shut down, the emulator to have exited, and a new process to launch the same installation without saved-memory resume. A Win95 safe-to-power-off screen may require the manager to complete emulator shutdown. Process identity, disk identity, and observed boot behavior belong in the evidence record.

5. Evidence model and limits of a pass

5.1 Guest checks and live qualification

`check.py` evaluates all three phases against the run manifest. It checks the selected suite, OS generation, identifiers, device state, functional results, and matching transition-request records. Missing, malformed, stale, or inconsistent reports fail the relevant phase. A zero exit status means the selected guest checks passed.

Every portable checker result retains `liveQualified: false`. Independent observations must still establish the exact VM and disk, selected USB routes, real restart and cold-boot completion, absence of new device/source prompts, and final cleanup. When networking is selected, that review also includes the DHCP lease and protocol binding. The current tool does not issue a machine-verifiable certification after that review.

Observation	What it supports	Additional evidence needed
Driver binding plus started status	Guest reports an active expected device node	Exact binary/version, correct ancestry, successful operation
Two in-window clicks	Windows delivered the expected pointer messages	Host input route identifying the USB tablet
Verified phase files	Selected filesystem path preserved tested bytes	USB attachment route and actual lifecycle transitions
DNS and HTTP response	Guest completed the configured network exchanges	Lease/binding evidence; broader application testing if claimed
Exit request record	Probe requested a particular transition	Observed completion and correct subsequent boot
Passing import audit	Required names are present in supplied exports	Loader, forwarding, initialization, and runtime behavior

5.2 Integrity and trust assumptions

Run identifiers and checksums help prevent accidental mixing of media, executables, and reports. A marker copied to another volume is still a matching marker. SHA-256 checksums without signatures do not authenticate an author, and guest-generated JSON is not secure remote attestation. The framework assumes an honest test operator and seeks reproducible diagnostics, not resistance to a malicious guest or a forged evidence package.

The report retains enough context to explain failures, including device descriptions and registry diagnostics. Before sharing an evidence bundle, the operator should inspect it for identifying environment details. That inspection should preserve the device identities and observations needed to assess the claimed result.

5.3 Failures remain useful results

A missing HID driver, problem code, wrong marker, corrupt seed, failed network exchange, or interrupted restart is an informative outcome. It should be published as a failure or incomplete run with the relevant evidence, rather than translated into a broad claim that all Win95 USB support is impossible.

Interrupted runs also require care: because the counter advances on execution, relaunching the probe can consume another phase without a boot transition. Archive the partial record, resolve the cause, and prepare a fresh run for qualification. Removal disables only this tester's startup entry; driver removal remains the driver's own procedure.

6. Recorded validation results

The following results describe the local release validation recorded on 13 September 2026. They are host-side software and artifact checks. No result in this table represents execution of the preview inside Windows.

Check	Recorded result	Interpretation
Portable automated tests	12 passed from the extracted archive	Preparation and report-checking behavior work in the tested host environment
Parent Perslis integration checks	8 passed	Host-side integration assumptions and rejection paths passed; no live VM coverage
Total host-side tests	20 passed	Sum of the two suites above
Archive rebuild	Byte-identical	Rebuilding the extracted source with the same environment reproduced the release ZIP
Archive checksums	Verified	Included files matched the archive's recorded hashes
Auxiliary media readback	Passed	Prepared floppy/FAT16 content and layout were checked from the host
Imported API names	All found	Static audit against preserved OSR2 DLL exports passed
Live guest execution	Not performed	No demonstrated install, input, storage, network, or boot result
Physical hardware	Not performed	No measured controller or peripheral compatibility

The portable tests include negative cases for incorrect run data, OS mismatch, inspection snapshots supplied as phase evidence, unsuitable device state, storage corruption, and invalid network responses. They also test that successful file results cannot pass the storage suite without the expected USB driver evidence. Native C and host-generated data patterns agree. Media checks inspect the partitioned FAT16 structure and installer floppy contents.

Host-side Perslis checks cover additional evidence and preflight failures, including a mismatch between the registered target and the proposed guest. These tests use controlled fixtures; describing them as live Windows integration tests would overstate their coverage.

The static API audit includes both ordinary executable imports and the named Configuration Manager functions resolved dynamically. Export presence is a useful compatibility screen, but it leaves loader behavior, forwarded exports, calling semantics, and initialization unproven. The reference profile therefore retains `runtimeQualified: false`.

6.1 Precisely bounded reproducibility

The archive result was obtained by extracting the preview and rebuilding it in the same recorded host/toolchain environment. It does not establish identical output from arbitrary compiler versions, operating systems, or packaging tools. New test media intentionally use a fresh run identifier and are not claimed to be byte-identical across independent preparations.

The public archive includes the 12 portable tests. The additional eight checks belong to the parent integration and are not required to use the portable kit. The release ledger records `guestTestExecuted: false`, `physicalHardwareTested: false`, and `published: false` alongside the successful host results.

7. Qualification roadmap: VM first, physical devices later

7.1 Complete the reference VM experiment

The next useful milestone is a complete storage-suite run on the stated OSR2/UHCI profile using separately obtained compatible drivers. Storage can proceed independently while HID remains unresolved. The operator should record driver provenance and versions, prepare fresh auxiliary media, and retain the same storage image across the three phases.

The candidate installation must be selected through its owning VM manager. Before launch, confirm the existing licensed disk, configuration, and available test slot. In this project's environment, only one guest or isolated test session is permitted at a time. An occupied slot is an operational prerequisite to resolve, not a reason to substitute another Windows release or bypass lifecycle ownership.

A reviewable result must connect the manifest and executable to the exact guest, show the storage image attached through UHCI USB, collect all phase and transition records, and document actual restart and cold launch. The run must end with tester startup removed and guest/helper cleanup verified. Driver/source prompts or recovery actions must be reported explicitly.

7.2 Resolve and qualify HID separately

A usable mouse/tablet result requires a compatible driver stack before the pointer test can pass meaningfully. Earlier Win98 HID backport experiments exposed unresolved kernel/VxD dependencies; they have not produced a qualified driver in this preview. The release excludes those donor binaries and experimental no-op compatibility patches.

Future driver work must preserve the semantics of required interfaces and demonstrate actual loading, enumeration, input delivery, and persistence across the same transitions. Import-name substitution alone is insufficient evidence. A first success should name the exact controller, tablet model, guest build, and driver versions, with any remaining restrictions.

7.3 Adapt the method for a physical machine

Physical testing is a future extension, not an already prepared raw-device installer. The current tools create image files and do not provision a real USB stick. Before hardware qualification, add an explicit media-preparation procedure and adapt the profile/checker for the actual controller and driver identities. Network checks also need adjustment if the machine does not use the reference NAT subnet.

A hardware record should identify the motherboard and BIOS settings, controller PCI ID, device VID/PID and revision, intervening hub if any, driver versions, guest build, and test-media identity. Begin with a dedicated test volume and a recoverable OS installation. Preserve an independent input method while testing HID.

For physical storage, repeat the marker, seed, write/read, restart, and power-cycle sequence while documenting actual device power state. A computer power cycle may leave a powered hub or peripheral energized; that distinction matters to the claim. Subsequent work can add hot-plug behavior, larger transfers, sustained operation, and additional devices. None is measured by the present 64 KiB per-phase test.

A compatibility matrix should grow one observed configuration at a time. A successful emulated controller/device pair would justify a VM-specific result; physical support requires its own evidence.

8. Reproduction and document evidence

8.1 Minimal storage experiment

Use the preview archive's instructions and an existing licensed OSR2 VM. Python 3.9+ and mtools are required for this media-preparation example. The archive includes the guest executable; rebuilding additionally requires an i686 MinGW-w64 compiler and a POSIX shell.

```
python3 prepare.py --out runs/storage-01 --suite storage \
  --storage-drive D --floppy
```

Choose the real guest drive letter before preparation. Through the VM manager, attach `U95TEST.IMG` as a read-only floppy and `USBTEST.IMG` read-write through the UHCI USB storage device. Complete the separately supplied driver's setup before starting the test. Then run inside Windows:

```
A:\U95TEST.EXE /install
C:\U95TEST\U95TEST.EXE /run
```

Follow the three-phase procedure in `docs/GUEST.TXT`, recording external transport and lifecycle evidence. After final shutdown, collect `C:\U95TEST` using the manager's existing transfer method or read-only offline extraction. Evaluate it on the host:

```
python3 check.py --run runs/storage-01 --evidence collected/U95TEST
```

Use `--suite input` for HID diagnostics, `--suite all` when both paths are available, and `--network` to add the PCnet/DNS/HTTP checks. `docs/TEST_REPORT.md` supplies the reporting template. A successful checker exit remains subject to the external review described in Section 5.

8.2 Artifact identity

This paper describes `win95-usb-lab-0.1.0-preview.1.zip`.

Release ZIP SHA-256

```
5a197c390a5e1f7ee3ead7b750f902ff5f52b06f4bd5e28fe9ac8131ee3a1dfb
```

Guest executable SHA-256

```
850a14b012f09fd1a05ff4110e1207a7aa9449b14ae8cceca5cd6815423de728
```

The companion white-paper folder includes `validation.json` and `import-audit.json`, copied from the local release evidence. The source basis is the named archive, especially `guest/probe.c`, `guest/storage.h`, `prepare.py`, `review.py`, `audit.py`, `package.py`, and `profiles/qemu-osr2.json`. The white paper is a separate document artifact and does not modify the identified release archive.

8.3 External references

- Conner McCoy and Rudolph Loew. XUSBSUPP: eXtended USB Supplement for Windows 95 OSR2. Authors' package documentation, 2014.
- QEMU Project. USB emulation. Official system-emulation documentation.
- Microsoft. CM_Get_DevNode_Status. API status and problem-code semantics.
- Microsoft. ExitWindowsEx. Shutdown-request semantics.

External references were consulted on 13 September 2026. They provide background and API descriptions; the project-specific validation claims come from the recorded artifacts above.