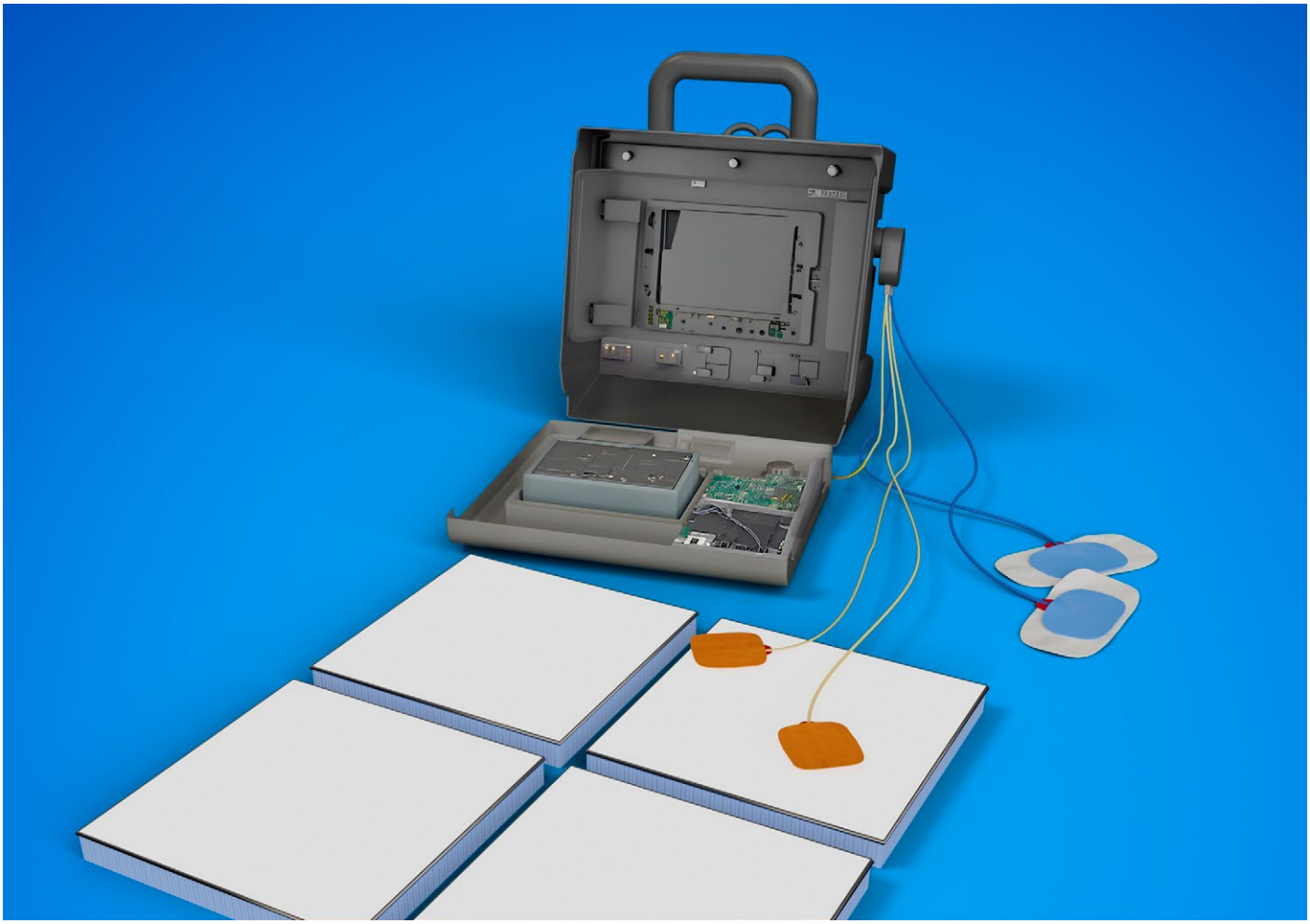


Bild: Moritz Reichartz

Werkzeugmacher

c’t-Notfall-Windows 2024: Tipps und Tricks zum Erweitern

Folgen Sie uns hinter die Kulissen des Bausatzes unseres Notfallsystems, den wir Anfang des Jahres vorgestellt haben. Wir zeigen, wie er tickt, und liefern darauf aufbauend Beispiele, wie Sie den Bausatz selbst erweitern können – Werkzeug hat man schließlich nie genug.

Von Peter Siering

Kurz zusammengefasst: Wir veröffentlichen das Notfall-Windows alljährlich als Bausatz, weil Microsoft das als Basis genutzte Minimal-Windows „Windows PE“ (Preinstallation Environment) zwar öffentlich zum Download bereitstellt, aber keine Lizenzen dafür ausgibt, die uns das Weiterverbreiten erlauben würden. Der Bausatz arbeitet dazu Skripte ab, die in einer Sprache verfasst sind, die das Programm WinBuilder einst eingeführt hat.

Die Basis unseres Notfallsystems bildet ein Projekt auf Grundlage von Windows PE. Das haben wir nicht selbst erfunden. Dritte haben es schon länger entwickelt. Seit vergangenem Jahr nutzen wir als Basis PhoenixPE. Das im Folgenden Erklärte gilt ebenso dafür. Das Ziel aller PE-Projekte ist, das im Funktionsumfang begrenzte Preinstallation Environment (PE) soweit aufzupeppeln, dass es reguläre Windows-Programme ausführt.

Ein typischer, offizieller Vertreter eines solchen PE-Systems ist die Umgebung, die von einem Windows-Installationsdatenträger startet, um das Betriebssystem startfähig auf die Festplatte eines PCs zu überspielen – daher kommt auch der Name „Preinstallation Environment“. Im „Windows Assessment and Deployment Kit“ (ADK) liefert Microsoft Werkzeuge, um diesen Prozess zu beeinflussen; frühe PE-Projekte griffen darauf zurück, heute geht es nahezu ohne.

Auf einem „Wiederherstellungslaufwerk“, das Sie sich unter Windows vom

c't kompakt

- Spezialisierte Skripte schrauben unser Notfallsystem auf Basis von PhoenixPE zusammen.
- Die Skripte reichern dazu Microsofts Preinstallation Environment mit weiteren Windows-Komponenten und zusätzlicher Software an.
- Die Skriptsprache ist schräg, lässt sich aber leicht erlernen, um das Notfallsystem zu erweitern oder anzupassen.

gleichnamigen Assistenten erstellen lassen können, findet sich ebenfalls eine PE-Variante. Der Windows-eigene Assistent beschreibt einen USB-Stick, von dem dann eine modifizierte PE-Umgebung startet. Die enthält aber vergleichsweise wenig Programme. Das einzig interaktive Element, um mal auf Dateien zu schauen, ist die Eingabeaufforderung.

Microsofts eigene PE-Umgebung geizt bei den enthaltenen Programmen, oft steht gerade mal eine Eingabeaufforderung zur Verfügung. Zudem fehlen viele Bibliotheken, deshalb läuft selbst so manche portable Software nicht, die ohne Installation auskäme. Und: In der 64-Bit-Umgebung stecken auch keine Funktionen, um 32-Bit-Programme auszuführen.

PE aufbrezeln

Die Kunst aller PE-Projekte besteht darin, das in der Datei Boot.wim steckende Minimal-Windows so weit aufzubrezeln, dass es weitere Programme ausführen lernt. Das erreichen die Projekte, indem sie das Basissystem um Registry-Schlüssel anreichern, die Software üblicherweise erwartet, nötige Bibliotheken hinzufügen und sogar das Subsystem ergänzen, um in einer 64-Bit-Umgebung, auch 32-Bit-Programme auszuführen (WOW64 [1]).

Dazu picken die Projekte in PE fehlende Dateien aus der install.wim-Datei einer Original-Windows-DVD heraus und pflanzen sie der Boot.wim ein. Zentrale Komponenten, die bei einer regulären Windows-Installation für einen geordneten Systemstart sorgen, ersetzen Sie mit eigenen Kreationen: pecmd.exe zündet die Stufen und bestückt unter anderem das Startmenü. Analog kümmert sich penetwork.exe ums Netzwerk.

Zu den Skripten: Der in Delphi (Borlands Pascal IDE) geschriebene, ursprünglich verwendete WinBuilder hatte diverse Macken und der Quelltext war nicht zugänglich. Mit PEBakery hat Hajin Jang Anfang 2018 auf GitHub eine Winbuilder-Alternative veröffentlicht, die er in C# implementiert und unter GPL-Lizenz gestellt hat. Das Programm hat sich zu einer großartigen Alternative entwickelt. Wir nutzen diese seit 2019.

PEBakery verhält sich, wo es darauf ankommt, komplett kompatibel zum WinBuilder und ist weitgehend frei von Macken. An bestehenden Skripten sind verhältnismäßig wenig Änderungen notwendig, um sie vom WinBuilder auf PEBakery umzustricken. Spezielle Optionen, die PEBakery bestimmte WinBuilder-Marotten nachbilden lassen, helfen obendrein.

Gesamtstruktur

Die grundsätzliche Struktur eines Projekts hat PEBakery beibehalten, also die Art und Weise, wie die Skripte zu einem größeren Ganzen zu bündeln sind: Ein Projekt besteht aus einem Verzeichnis unterhalb von „Projects“ (etwa Projects\PhoenixPE) mit einer Datei namens „script.project“ darin, die das Gesamtprojekt beschreibt, etwa das c't-Notfall-Windows 2024.

Alle weiteren Skripte liegen in diesem Verzeichnis oder in Unterverzeichnissen. PEBakery liest diesen Projektbaum beim Start vollständig ein und führt die in script.project enthaltenen Anweisungen aus. Im c't-Notfall-Windows 2024 sorgen diese dafür, dass die Willkommensnachricht erscheint und dem Nutzer Hinweise zur Bedienung gibt.

Zwei weitere Dateien sind wichtig: Im Hauptverzeichnis des Gesamtprojekts (etwa c:\ctnot) liegt die Datei PEBakery.ini. Sie beschreibt den Namen des Projekts beziehungsweise den Namen des Unterverzeichnisses (PhoenixPE) und einige weitere Startoptionen für PEBakery.

Die zweite Datei, die im Pfad c:\ctnot\Projects\PhoenixPE liegt, heißt folder.project. Sie beschreibt weitere Ordner, die zum Projekt gehören sollen und unterhalb von c:\ctnot\Projects liegen müssen – PhoenixPE schlägt hier MyApps vor, um dort eigene Skripte abzulegen, die das Projekt erweitern.

Skriptkern

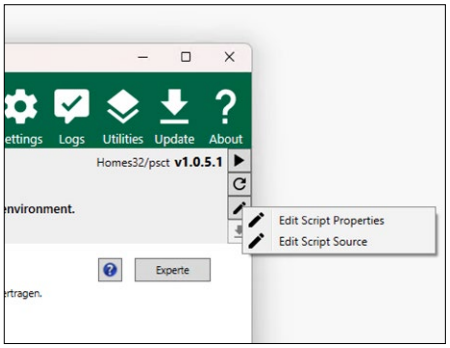
Auf den ersten Blick ähneln Skripte INI-Dateien, weil Sie in eckigen Klammern gesetzte Abschnitte zur Gliederung verwenden und Variablen sowie Optionen das typische Format einer Zuweisung verwenden. In der Tat machen sich unter anderem unsere Erweiterungen diese Ähnlichkeit auch zunutze, indem Sie Funktionen zum Bearbeiten von INI-Dateien auf Skripte loslassen, um darin Anpassungen vorzunehmen.

Der Abschnitt `[Main]` beschreibt die Grunddaten eines Skripts: Dort steht, wie es heißt (`Title`), was es tut (`Description`), wer es geschrieben hat (`Author`), wann es laufen soll (`Level`), ob es laufen soll (`Selected`), ob es laufen muss (`Mandatory`), welche Version es ist (`Version`) und wann es geschrieben wurde (`Date`). Manche Zeilen sind Konvention, andere wichtig fürs Gesamtprojekt. Das Folgende ordnet das bei Bedarf ein.

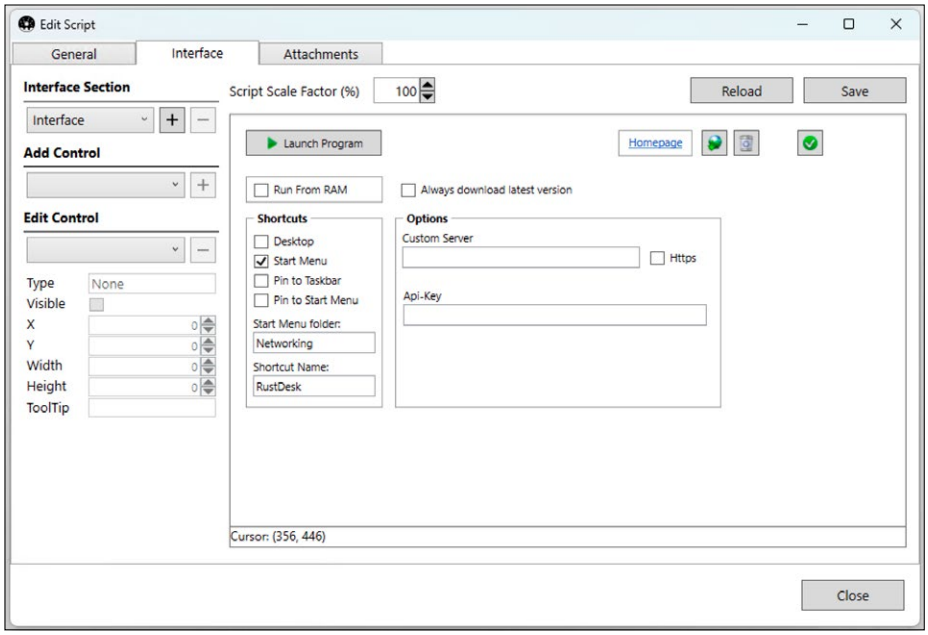
Im Abschnitt `[Variables]` werden Variablen vorbelegt. Das ist immer dann sinnvoll, wenn ein Skript Daten mehrfach verwendet, die sich ändern können, zum Beispiel Download-URLs. Variablen erkennen Sie in den Skripten daran, dass ihr Name mit einem Prozentzeichen beginnt und endet – sie ähneln Variablen in Batch-Dateien, teilen aber keinen Namensraum, kommen sich also nicht ins Gehege.

Sowohl PEBakery als auch WinBuilder kennen einen Satz vordefinierter Variablen: In %BaseDir% steht der Name des Wurzelverzeichnisses, zum Beispiel c:\ctnot, %ProjectDir% verweist auf die Projektdateien, etwa c:\ctnot\Projects\PhoenixPE. Über %ScriptFile% finden Skripte den eigenen Dateinamen heraus.

Die vorgenannten Variablen und einige mehr kann ein Skript nicht verändern, sie heißen auch „Fixed Variables“. PEBakery definiert sie beim Projekt- oder Skriptstart. Solche Variablen sind wie auch im Abschnitt `[Variables]` in der script.pro-



Über das Stiftsymbol lassen sich aus PEBakery heraus, Skripte bearbeiten. Für Vielschreiber ist es hilfreich Visual-Studio Code als Editor zu verwenden.



Um die Bedienoberfläche für ein Skript zu erstellen, bietet PEBakery einen eigenen grafischen Editor, um Elemente wie Eingabefelder zu positionieren und das Aussehen komfortabel zu gestalten.

ject-Datei definiert immer in allen Skripten sichtbar („Global Variables"). Variablen in regulären Skripten sieht nur das jeweilige Skript.

Ein Skript muss Variablen nicht zwangsläufig im Abschnitt `[Variables]` definieren. Es kann das auch jederzeit über den Aufruf `Set,%MeineVariable%,"Mein Text"` tun – das Beispiel belegt die Variable mit „Mein Text" als Inhalt. Wenn das Skript an den Aufruf die Option `GLOBAL` mit einem Komma anhängt, dann können auch andere Skripte auf den Inhalt zugreifen und die Variable verändern. Eine übersichtliche Liste der Variablen liefert PEBakery in den Logs.

Die Handlungsanweisungen, die ein Skript ausführt, landen in einem dritten Abschnitt: `[Process]`. Hier kann ein Skript in beliebiger Weise die verschiedenen Befehle kombinieren, die PEBakery bereitstellt: Archive herunterladen, entpacken, Dateien kopieren, Textdateien manipulieren und externe Programme ausführen.

Skripte können die Registry des entstehenden Notfallsystems manipulieren, Bedingungen prüfen und passende Skriptpfade ausführen, WIM-Dateien bearbeiten, in Grenzen rechnen, Listen bearbeiten und in Schleifen durchexerzieren. In umfangreichen Skripten sollte ein Autor weitere Abschnitte einführen, die ein Skript wie ein Unterprogramm aufrufen kann.

In vielen Skripten findet sich ein weiterer Abschnitt, nämlich `[Interface]`. Er beschreibt die Bedienoberfläche, die vom Nutzer wählbare Optionen darstellt. Einige Skripte bauen daraus mehrseitige Dialoge, um dort Details festzulegen; viele kommen ganz ohne aus. Niemand muss für die Oberfläche kryptische Texte verfassen: Ein in PEBakery enthaltener interaktiver Editor für die Bedienelemente hilft dabei.

Es gibt noch einige weitere Abschnitte, die beim Bearbeiten von Skripten in PEBakery entstehen, um beispielsweise Logos für das Skript zu verwalten und zu speichern. Es ist sogar möglich, Binärdateien in ein Skript hochzuladen, es wird dann als Text Base64-kodiert als Abschnitt in die reine Datei eingefügt (ähnlich wie Attachments in E-Mails) – das Skript kann mit speziellen Befehlen hinzugefügte Binärdaten extrahieren.

Skriptrang

Die Reihenfolge, in der PEBakery einen vollständigen Bausatz ausführt, hängt von zwei Faktoren ab: Die grobe Reihenfolge gibt „Level" im `[Main]`-Abschnitt vor. Im Level 1 bis 4 laufen Vorbereitungen für das Grundsystem, in Level 5 ergänzt ein Projekt üblicherweise Programme, in Level 6 fügt es Treiber hinzu, von Level 7 bis 9 wird aufgeräumt und schließlich die finale ISO-Datei gebaut. Auf Level 0 können Entwickler Skripte deponieren, die nicht laufen sollen und die PEBakery nicht anzeigt, deren Funktionen aber andere Skripte aufrufen können.

Es hat sich eingebürgert, innerhalb der Level den einzelnen Skriptdatei- und Verzeichnisnamen dreistellige Zahlen voranzustellen. Die Sortierung gibt die Reihenfolge vor, in der PEBakery die Skripte ablaufen lässt. Großzügige Abstände erleichtern es, eigene Skripte dazwischenzuschieben.

Grundsätzlich kann ein Skript im Gesamtkontext eines Projekts laufen, wenn der Benutzer den Build-Knopf im PEBakery-Fenster betätigt. Alternativ lässt sich ein einzelnes Skript über den Play-Knopf direkt starten. PEBakery führt das Skript dann auch unabhängig davon aus, ob es im Abschnitt `[Main]` als „Selected" markiert ist oder nicht.

Der Start eines einzelnen Skripts erleichtert das Testen erheblich, weil bis zum Ausführen des frisch ergänzten Codes nicht immer das ganze Projekt laufen muss. Aber: Skripte verändern Dateien im Projektbaum, laden sie herunter, packen sie aus und modifizieren sie. Ein Skript sollte deshalb, wenn es einzeln läuft, auch immer im Gesamtkontext getestet werden. Und das in zwei Situationen: einmal in einem sauberen Projektbaum und einmal in einem bereits benutzten.

Ein sauberes Skript sollte sich nicht auf einen bestimmten Zustand verlassen, sondern immer sicherstellen, dass er besteht: Die meisten Skripte prüfen zunächst deswegen, ob eine Datei schon heruntergeladen ist, bevor sie diese entpacken. Es kommt aber vor, dass unvorhergesehene Dinge passieren: Reißt ein Download ab, könnte der Entpackschritt fehlschlagen, weil die Datei unvollständig ist. Für solche Situationen sind die wenigsten Skripte gewappnet und nehmen in Kauf, dass dann der Projektlauf mit einem Fehler abbricht – eine technisch mögliche Absicherung der Downloads per Prüfsumme erfolgt aus pragmatischen Gründen nicht.

PhoenixPE und somit das c't-Notfall-Windows versuchen bei einem vollständigen Projektlauf, die wesentlichen Log-Dateien zu einem komprimierten Paket zu schnüren. Das lässt sich bei anhaltend fehlschlagenden Bauversuchen im Unterverzeichnis „log" per E-Mail für eine Supportanfrage verschicken.

Läuft nur ein einzelnes Skript, kommt der Log-Viewer von PEBakery ins Spiel. Er zeigt auf Wunsch in HTML-Form die Ausgaben eines Skripts an. So können Sie den Ablauf eines selbst entwickelten Skriptes nachträglich verfolgen, aber ebenso gut studieren, wie sich andere Skripte verhal-

ten. Letzteres ist essenziell für den Einstieg in die Skripterei: Was Sie aus funktionierenden Skripten abschreiben können, brauchen Sie sich nicht selbst auszudenken.

Skriptgerüst

PhoenixPE bringt obendrein Hilfen mit, ein Gerüst für typische Skripte zu generieren. Sie finden diese im Projektbaum unter PhoenixPE/Toolbox/ScriptFactory. Um ein Skriptgerüst zu erzeugen, reicht es, die Felder auszufüllen und den Knopf „Create Script“ zu betätigen. Die Varianten der Skripte sind selbst erklärend, ebenso die meisten Felder. Ein Hinweis zu „Script Folder“: Das Feld steuert, ob das Skript zusätzlich unterhalb des per Level vorgegebenen Verzeichnisses im Projektbaum in ein weiteres Unterverzeichnis gesteckt wird (wie sie in Applications existieren).

PEBakery ordnet ein frisch erstelltes Skriptgerüst direkt in den Projektbaum ein. Sollten Sie sich beim Anlegen vertan haben, müssen Sie durch die Aktion angelegte Dateien und Verzeichnisse im Dateisystem selbst wegräumen. Innerhalb von PEBakery gibt es dafür keine Funktionen. PEBakery bemerkt solche Änderungen nicht von sich aus. Sie müssen per Klick auf den Refresh-Knopf in der oberen Werkzeugleiste nachhelfen. Das Gleiche gilt für den Fall, dass Sie Ihre Skripte direkt im Dateisystem ergänzen.

Sämtliche Bearbeitungsschritte für ein Skript sind aus PEBakery heraus zugänglich, entweder auf der Skript-Seite oder per Kontextmenü im Projektbaum. PEBakery selbst liefert Funktionen zum Bearbeiten der Skripteigenschaften („Edit Script Properties“), lädt etwa Logos hoch, setzt Optionen aus dem Abschnitt `[Main]`, ordnet Bedienelemente fürs Skript an und verwaltet Dateianhänge in Ordnern.

Zum Bearbeiten der Quelltexte („Edit Script Source“) spannt PEBakery Notepad ein, wenn kein Visual Studio Code installiert ist. VS Code ist sehr zu empfehlen, denn es gibt sogar eine Erweiterung für PEBakery-Skripte, die nicht nur die Sprachelemente farblich aufhübscht, sondern beim Schreiben von Skripten auch gleich mit Vorschlägen hilft; so lernt sich die gewöhnungsbedürftige Syntax bequemer.

Skriptbeispiel

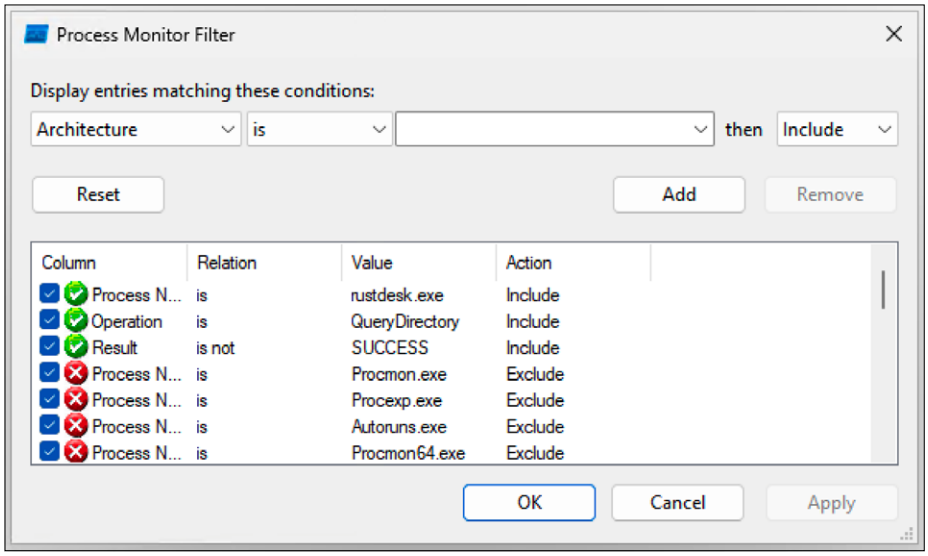
Das folgende konkrete Beispiel beschreibt die Schritte zur Integration von RustDesk (einer Alternative zu AnyDesk). Bevor Sie sich ans Werk machen, sollten Sie ausprobieren, wie sich die Software verhält, wenn Sie diese manuell ins Notfallsystem bringen: Kopieren Sie die EXE-Datei in ein neues Verzeichnis auf einen mit dem Bausatz erzeugten Stick, booten Sie den und starten Sie das Programm: RustDesk.exe meckert, weil ihm DLLs fehlen. Den Fehlermeldungen nach fehlen glu32.dll und opengl32.dll.

```
[Main]
Title=RustDesk
Author=ps@ct.de
Level=5
Version=1.1.0.0
Description=Remote desktop software and open source TeamViewer alternative
Date=2023-11-04
Selected=False
Mandatory=False

[Variables]
%ProgramFolder%="RustDesk"
%ProgramExe%="rustdesk.exe"
%DownloadURL%=https://github.com/rustdesk/rustdesk/releases/download/1.2.3/
rustd
%ConfigFile%="RustDesk2.toml"

[Process]
Echo,"Processing %ScriptTitle%..."
If,%cb_RunFromRam%,Equal,True,RunFromRam
If,%cb_AlwaysDownload%,Equal,True,DirDeleteEx,"%ProgramsCache%\%ProgramFolder%"
If,Not,ExistFile,"%ProgramsCache%\%ProgramFolder%\%ProgramExe%",Run,%ScriptFile%
FileCopy,"%ProgramsCache%\%ProgramFolder%\%ProgramExe%","%TargetPrograms%\%Progr

RequireFileEx,AppendList,\Windows\System32\sas.dll
RequireFileEx,AppendList,\Windows\SysWOW64\sas.dll
RequireFileEx,AppendList,\Windows\System32\opengl32.dll
RequireFileEx,AppendList,\Windows\SysWOW64\opengl32.dll
RequireFileEx,AppendList,\Windows\System32\glu32.dll
RequireFileEx,AppendList,\Windows\SysWOW64\glu32.dll
RequireFileEx,ExtractList

[DownloadProgram]
Echo,"Downloading %ScriptTitle%..."
WebDownload,%DownloadURL%,"%ProgramsCache%\%ProgramFolder%\%ProgramExe%",NOERR
If,Not,#r,Equal,200,Halt,"Download failed: The code returned was [#r]."
```

Das hier nur abgespeckt und nicht in voller Breite wiedergegebene Skript zur Integration von RustDesk steckt bereits im Bausatz unseres Notfallsystems.

Diese DLLs aus dem Lieferumfang von Windows können Sie nun versuchsweise ins Notfallsystem kopieren. Idealerweise bedienen Sie sich aus dem Dateifundus einer Windows-Installation mit identischer Version wie der, die Sie an den Bausatz verfüttern. Es genügt, diese DLL-Dateien in das Verzeichnis zu kopieren, in dem auch die heruntergeladene RustDesk-Datei liegt – das Programm ist erfreulicherweise nicht kompliziert in einem Installer verpackt.

Trotz der beiden ergänzten DLLs wird RustDesk nicht starten. Es gibt jedenfalls keinen Mucks von sich. Hier hilft es dann nur, tiefer in die Ursachenforschung einzusteigen: Dafür eignet sich das Sysinternals-Werkzeug Procmon gut. Es zeichnet die Funktionsaufrufe, die ein Programm tätigt, auf und zeigt sie an. Procmon können Sie im laufenden Notfallsystem herunterladen und starten – es braucht selbst keine Installation und alle von Procmon benötigten Bibliotheken stecken im Notfallsystem. Eine Serie von Einführungs-



Mit einem Filter im Sysinternals Processmonitor findet man schnell heraus, welche DLLs ein Programm laden möchte, aber nicht findet und auch nicht lauthals reklamiert.

artikeln zu Procmon haben wir in [2, 3, 4] veröffentlicht, hier die Kurzfassung.

Schalten Sie nach dem Start von Procmon unbedingt mit Strg+E die automatisch startende Aufzeichnung von Ereignissen ab – es speichert alles im RAM, und darin geht schnell der Platz aus. Erst kurz bevor sie einen weiteren Startversuch von RustDesk unternehmen, lassen Sie Procmon wieder die Ereignisse aufzeichnen. Nach dem Start beendeten Sie diese schnell wieder.

Konstruieren Sie nun einen Filter in Procmon, um die angezeigten Ergebnisse einzugrenzen. Bei uns hat sich bewährt, den „Process Namen" auf das untersuchte Programme zu beschränken. RustDesk tritt hier in zwei Varianten an, unter dem Originalnamen mit langer Versionssignatur und schlicht als rustdesk.exe. Der letzte Name ist der hilfreiche. Als weitere Filterkriterien verwenden wir „Operation" und dabei die Funktion „QueryDirectory" und schauen uns nur fehlgeschlagene Aufrufe an („Result is not SUCCESS").

Mit den genannten Kriterien finden Sie sehr schnell von einem Programm verwendete DLL-Dateien, die es benötigt, nicht findet, aber auch nicht selbst in einer Fehlermeldung konkret benennt – im Fall von RustDesk ist das die Datei sas.dll. Wenn Sie diese DLL neben den zuvor schon genannten in das Verzeichnis werfen, in dem RustDesk liegt, startet die Software im Notfallsystem und lässt sich rudimentär ausprobieren (um sicherzustellen, dass nicht noch weitere Dateien fehlen).

Sie könnten nun diese DLLs als Dateien einfach als Anhang in dem Skript aufnehmen, aber das ist aus mehreren Gründen doof: Zum einen stellt das eine Lizenzverletzung dar; Sie dürfen nicht einfach Dateien aus dem Windows-Lieferumfang einbauen – jedenfalls, wenn Sie den Bausatz beziehungsweise Ihr Skript auch anderen zur Verfügung stellen möchten. Zum anderen müssten Sie dann selbst sicherstellen, dass die aus einer zum gebauten Notfallsystem passenden Windows-Version stammen.

Verpackungskünstler

Der Einbau von RustDesk über ein Skript ist dann doch etwas aufwendiger: Ausgehend von einem generierten Skriptgerüst mit der Vorlage „Simple Download App" öffnen Sie das Skript am besten im Editor, etwa Visual Studio und gehen es Zeile für Zeile durch. Alle Stellen, an denen Sie sich zu schaffen machen sollten, sind mit einem Kommentar `// TODO` versehen, etwa den Namen für den Ordner, den der Bausatz für das Programm anlegen soll.

Sie werden dann schnell über eine Gemeinsamkeit vieler Skripte stolpern: Sie behandeln 32- und 64-Bit-Programmversionen separat (die `TODO`s fordern Dateinamen und Download-URLs für die jeweilige Version). Das hat oft historische Gründe. Grundsätzlich würde es genügen, die 32-Bit-Version eines Programms ins Notfallsystem einzubauen, weil sie auch in der 64-Bit-Ausgabe des Systems funktionieren wird (anders als in einem regulären PE); die 64-Bit-Ausgabe integriert schließlich Windows-32-On-Windows-64 (WOW64) ins Notfallsystem.

Mit dem aktuellen Bausatz haben wir uns allerdings entschlossen, nur noch 64-Bit-Versionen zu unterstützen. Das im Folgenden beschriebene RustDesk-Skript lädt deshalb nur die 64-Bit-Programmversion herunter und nimmt keine Rücksicht auf eventuelle 32-Bit-Eigenarten – mit einer Ausnahme: Sie ergänzt stets auch DLLs, die für 32-Bit-Programme nötig wären – schließlich ist eben nicht jedes Programm im ganzen System auch eine 64-Bit-Ausgabe; ohne diesen Kniff würden 32-Bit-Programmbestandteile streiken, wenn sie die 32-Bit-DLLs nicht finden.

Zeilen, um die von RustDesk vermissten DLLs ins Notfallsystem zu integrieren, sind im generierten Skript nicht enthalten. Fügen Sie dazu im Abschnitt `[Process]` zwischen den Kommentaren „Extract" und „Settings" passende Zeilen ein (siehe Listingkasten). Die sorgen dafür, dass beim Bauen die genannten DLLs aus der verwendeten Windows-Quelle genommen und in das Projekt eingebaut werden.

Die Aufrufe von `RequireFileEx,AppendList` fügen Sie einer Liste zur Bearbeitung hinzu, der letzte Aufruf mit `ExtractList` als zweitem Parameter erledigt den Einbau. Auf diese Weise „ziehen" viele Skripte Windows-Original-Dateien beim Bauen ins Notfallsystem. Die Befehle sind als Makros im Projekt definiert und rufen im Hintergrund Funktionen auf, die WIM-Dateien bearbeiten.

Um das Skript für die Veröffentlichung schön zu machen, können Sie noch ein zum Programm passendes Logo einbauen. Wir haben kurzerhand eines aus dem GitHub-Repository genommen. Klicken Sie zum Einbinden das Stiftsymbol rechts im Kopf des Skriptes an und rufen Sie die Funktion „Edit Script Properties" auf. Im unteren Bereich des dann aufgehenden Fensters können Sie eine passende Datei auswählen und so in das Skript aufnehmen lassen.

Die hier nicht vollständig wiedergegebenen Zeilen der Skriptdatei befassen sich damit, eine Konfigurationsdatei zu erstellen, die RustDesk den Weg zu einem privaten Vermittlungsserver weist. Dessen Daten können Nutzer in PEBakery eingeben. Für die Integration von RustDesk sind andere für ein Skript typische Aktionen nicht nötig, etwa das Erzeugen und Setzen von Schlüsseln in der Registry des Notfallsystems.

Wenn Sie eine Skript-Datei testen wollen, sollten Sie das zunächst in einem Bauverzeichnis tun, das bereits die Dateien eines erfolgreichen Laufs beinhaltet. So ist sichergestellt, dass ein Skript im Kontext korrekt arbeitet. Wenn Sie das überprüft haben, sollte im zweiten Schritt ein Test mit einem frischen Bauverzeichnis folgen – so gewährleisten sie, dass Ihr Skript auch auf anderen Rechnern funktioniert.

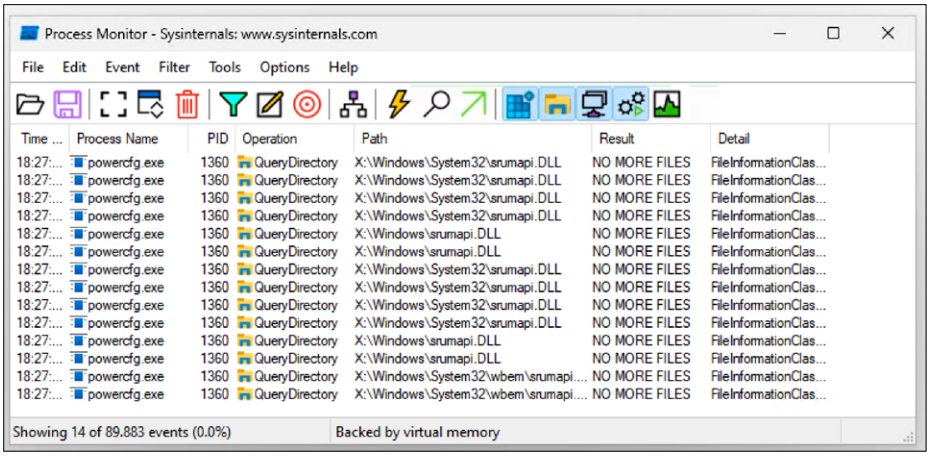
Detektivarbeit

Vom Handwerklichen abgesehen besteht die Kunst, eigene Skripte in den Bausatz zu integrieren, vor allem in Detektivarbeit: In hartnäckigen Fällen hilft Procmon aus den Sysinternals-Werkzeugen: Lassen Sie Procmon im Notfallsystem das betrachtete Programm beobachten. Schauen Sie sich zunächst fehlgeschlagene Operationen an. Oft liefern diese Fehlschläge Hinweise auf weitere fehlende DLLs und eventuell zu ergänzende Registry-Schlüssel. Hier kommt es nicht auf Vollständigkeit an, sondern vor allem, die wirklich nötigen Teile Schritt für Schritt zu identifizieren.

Der letzte Schritt zu einem perfekten Skript besteht dann darin, die gewonnenen Erkenntnisse anzuwenden: Sie müssen Wege finden, aus dem Download eines Installationspakets die Dateien herauszulösen. Die diversen Skripte in PhoenixPE und im Notfall-Windows verwenden dazu einen ganzen Satz von Spezialprogrammen, falls Werkzeuge wie 7-Zip scheitern; für einige bringt PhoenixPE sogar spezialisierte Makros mit.

Behalten Sie bei all dem im Auge, unter welcher Lizenz die Software steht, die Sie ins Notfallsystem einbauen. Freie Software, die ohne Einschränkung weiterverbreitet werden darf, können Sie wie auch andere Binär- oder Konfigurationsdateien als Dateianhang an ein Skript anhängen. Das allerdings erschwert Nutzern das Updaten von Programmversionen. Es ergibt also allenfalls bei zeitloser Software überhaupt Sinn.

Ein Hinweis noch zur Integration neuer Programme in die Menüstruktur des c't-Notfall-Windows: Die in PEBakery erzeugten Skriptgerüste sehen dafür die PhoenixPE-eigenen Methoden vor. Wir verwenden einen eigenen Ansatz, um die Menüs des Notfallsystems übersichtlicher zu gestalten. Die Datei pecmd_links.ini, die Sie im Unterverzeichnis Custom im Bauverzeichnis respektive im Bausatz-Zip-Archiv finden, beschreibt die Menüeinträge. Ergänzen Sie Ihre Skripte analog zu den vorhandenen Einträgen.



Filter im Processmonitor helfen, die relevanten Daten zu isolieren, etwa fehlschlagende Versuche von `powercfg /batteryreport`, weitere DLLs zu laden.

Die Menüeinträge sind Teil der Konfiguration für das eingangs schon erwähnte pecmd.exe. Das Programm startet die Bedienoberfläche des Notfallsystems. In seiner vollständigen Konfigurationsdatei landen die Einträge aus pecmd_links.ini und pecmd_pins.ini (für ans Startmenü angepinnte Programme, ebenfalls aus dem Custom-Verzeichnis) kombiniert mit während des Bauens generierten Einträgen. Um zum Beispiel beim Start des Notfallsystems automatisch Programme auszuführen, ist pecmd.ini die richtige Adresse, eine Zeile mit `EXEC=` genügt.

Für die Bearbeitung von pecmd.ini gibt es mehrere Möglichkeiten. Am einfachsten gelingt sie, indem Sie sie im Projektbaum in PEBakery links unter Shell PECMD auswählen und dann eine benutzerdefinierte Datei hinzufügen. Ein Beispiel dafür spuckt der Knopf „Sample Config“ aus. Allzu viel Aufmerksamkeit sollten Sie den Mechanismen aber nicht widmen: Der PhoenixPE-Entwickler Jonathan Holmgren krempelt den Bereich gerade um und will weg vom als Closed Source entwickelten pecmd.exe.

Wenn Sie die künftige Registry des Notfallsystems bearbeiten wollen, gibt es dabei etwas Wichtiges zu Beachten: Der Bausatz bindet Teile dieser vorübergehend in die Registry des Bausystems ein. Es ist wichtig, den richtigen Zweig anzusprechen, also den dazu eingebundenen Teilast. Und: Anders als in einem laufenden Windows ist dann zum Beispiel nicht CurrentControlSet gefragt, sondern ControlSet001. Nützlich kann das zum Beispiel sein, um den Suchpfad (PATH) im Notfallsystem für eigenhändig ergänzte Programme zu erweitern.

Harte Nüsse

Ein Tipp noch zur vom Winbuilder eingeführten „Sprache“ der Skripte: Die ist gewöhnungsbedürftig. So lässt sie Funktionen für Selbstverständlichkeiten in anderen Skriptsprachen wie zeilenweises Lesen von Textdateien vermissen. Die Skripte des Bausatzes enthalten dazu kreative Lösungen und bemühen im Zweifelsfall externe Skripte, etwa Batch-Dateien, PowerShell oder andere Werkzeuge. Eine Sprachreferenz finden Sie auf GitHub in den PEBakery-Repositories (siehe ct.de/ydzn).

Eine letzte Empfehlung: Tasten Sie sich langsam heran. Portable Software lässt sich leicht integrieren. Komplexere Anwendungen, die Treiber oder eigene Dienste mitbringen oder auf solchen aufbauen, sind schwer in die PE-Welt zu übertragen – nicht immer gelingt es. Unsere To-do-Liste umfasst unter anderem Bedienhilfen für Blinde, die wir bis jetzt nicht zum Laufen gebracht haben. Über sachdienliche Hinweise oder fertige Skripte freuen wir uns sehr. *(ps@ct.de)* c't

Literatur

[1] Axel Vahldiek, Altes im Neuen,32-Bit-Anwendungen unter 64-Bit-Windows, c't 23/2019, S. 150
[2] Axel Vahldiek, Unter dem Mikroskop, Windows analysieren mit dem Process Monitor – Teil 1, c't 16/2017, S. 148
[3] Axel Vahldiek, Schärfer stellen, Windows analysieren mit dem Process Monitor – Teil 2, c't 17/2017, S. 154
[4] Hajo Schulz, Noch mehr Durchblick, Windows analysieren mit dem Process Monitor – Teil 3, c't 18/2017, S. 162

Projektseite mit Downloads, Forum und Artikelverweisen: **ct.de/ydzn**