# 針對 OpenWrt/prplOS 架構之動態可展開式軟體視覺化可行性研究與實作策略報告

## 執行摘要

本研究報告旨在深入探討並回應關於在 OpenWrt 及 prplOS 嵌入式作業系統環境下，如何實現具備「可展開式（Expandable）」特性的軟體架構視覺化方案。針對使用者提出的需求——即將高階模組依賴關係與低階內部細節（如有限狀態機 FSM、API 呼叫）結合於單一互動式圖表中，並利用 GDB 與 eBPF 收集之數據驅動該視覺化——本報告進行了詳盡的技術可行性分析與架構設計。

研究結果顯示，在 OpenWrt/prplOS 特有的基於匯流排（Bus-centric）與服務導向架構（Service-Oriented Architecture, SOA）中，實現此類視覺化不僅可行，更是解決複雜系統可觀測性（Observability）問題的關鍵路徑。然而，現有的現成工具（Off-the-shelf tools）如 AppMap、Sourcetrail 或 Cytoscape.js，均無法在不經客製化的情況下直接對接 C 語言編寫且依賴 ubus 通訊的嵌入式系統。AppMap 提供了最符合「程式碼地圖（Code Map）」概念的使用者介面與互動邏輯，但缺乏對 C/C++ 及 ubus 的原生支援；Cytoscape.js 提供了強大的圖論繪製引擎與複合節點（Compound Nodes）功能，適合構建完全客製化的儀表板；而 Sourcetrail 則因其靜態分析的本質，無法有效捕捉執行時期的動態依賴與狀態變遷。

本報告提出了一套名為「prplVis」的資料轉換架構，建議採取「資料正規化（Data Normalization）」策略：利用靜態分析（ODL 解析）構建系統骨架，並結合動態追蹤（eBPF/GDB）填充行為血肉，最終將異質數據轉換為 AppMap 的開放 JSON 格式或 Cytoscape 的圖形結構。此策略能精確地將 prplMesh 等複雜組件的 FSM 狀態映射為可視化節點，並將 IPC 通訊具象化為模組間的連線，從而達成使用者手繪草圖中的互動式分析目標。

## 1. 緒論與問題定義

在現代嵌入式系統開發中，隨著軟體複雜度的指數級增長，傳統的靜態架構圖與線性的日誌分析已難以滿足系統理解與除錯的需求。特別是在 OpenWrt 與 prplOS 這類廣泛應用於網通設備的作業系統中，系統行為由數十甚至數百個獨立的守護行程（Daemons）與外掛模組（Plugins）透過非同步的行程間通訊（IPC）機制交織而成。使用者提出的核心需求在於建立一種「動態且可展開」的視覺化模型，該模型需具備以下特徵：首先是層次化（Hierarchical），允許從系統視角俯瞰模組關係，並能深入鑽取（Drill-down）至模組內部的狀態機與函數；其次是資料驅動（Data-Driven），圖表需基於實際執行數據（Execution Traces）而非設計文檔生成；最後是關聯性（Connected），需清晰呈現模組間的互動路徑。

### 1.1 OpenWrt/prplOS 的可觀測性挑戰

要評估此視覺化方案的可行性，首要任務是解構目標系統的架構特性，這些特性直接決定了數據收集與視覺呈現的邊界條件。

OpenWrt 與 prplOS 的架構本質上是鬆散耦合的。不同於單體式（Monolithic）應用程式，其「相依性」並非主要由編譯時期的符號連結（Symbol Linking）決定，而是由執行時期的 ubus 訊息傳遞定義。例如，一個管理 Wi-Fi 的 hostapd 行程可能依賴於 netifd 網路介面管理行程，這種依賴關係僅在系統運作時透過 ubus call 體現。因此，任何試圖僅通過靜態原始碼分析（Static Analysis）來構建架構圖的嘗試，都將遺漏最關鍵的互動邏輯。這正是 eBPF（Extended Berkeley Packet Filter）與 GDB（GNU Debugger）等動態分析工具介入的必要性所在。

此外，prplOS 引入了 Ambiorix 框架，這是一個高度抽象化的資料模型管理層。在 Ambiorix 中，軟體功能被封裝為「資料模型物件（Data Model Objects）」，其定義描述於 ODL（Object Definition Language）檔案中 1。這意味著，視覺化工具眼中的「模組」，在實作層面上可能對應於一個 ODL 物件、一個 so 動態連結庫，或是一個獨立的 Process。如何將這些異質的實體映射為統一的圖形節點，是本研究需解決的核心抽象問題。

### 1.2 「可展開式」視覺化的定義與需求

使用者手繪圖中展示的「可展開式」概念，在資訊視覺化領域對應於「複合圖（Compound Graph）」或「巢狀圖（Nested Graph）」理論。在這種圖形結構中，節點（Node）不再是原子化的實體，而是可以包含其他節點的容器（Container）。

針對使用者的具體需求，這種展開行為具有明確的語意：

* **收合狀態（Collapsed）：** 顯示系統的高階拓撲。節點代表主要的子系統（如 prplMesh Agent、Firewall），邊（Edge）代表跨行程的 API 呼叫或事件流。
* **展開狀態（Expanded）：** 顯示特定模組的內部構造。原本的單一節點展開為一個子圖（Subgraph），內部節點代表 FSM 的狀態（如 IDLE、CONNECTING）或內部 C++ 類別，內部的邊則代表狀態變遷（State Transitions）或函數呼叫堆疊。

實現此效果的關鍵在於：工具必須支援「多層次抽象」，並且能夠處理從動態數據中重建這種層次結構。這不僅是 UI 呈現的問題，更是資料結構設計的挑戰。

## 2. 系統架構分析與資料映射策略

為了將 GDB 和 eBPF 收集到的原始二進位數據轉換為使用者期望的視覺化圖形，我們必須建立一套嚴謹的資料映射（Data Mapping）邏輯。這需要深入理解 OpenWrt/prplOS 的底層機制，特別是 ubus 通訊協定與 FSM 的實作模式。

### 2.1 靜態骨架提取：從 ODL 到圖形節點

在繪製任何連線之前，必須先確立圖形中的「節點」為何。僅依賴動態追蹤（Tracing）會導致「倖存者偏差」，即只能看到被執行過的程式碼路徑，而無法呈現系統的完整全貌。因此，解析 Ambiorix 的 ODL 檔案是構建系統靜態骨架的必要步驟。

ODL 檔案定義了模組的資料結構與介面 1。利用 libamxo 提供的解析器或其 Python 綁定 1，我們可以遍歷 /etc/amx/ 目錄下的所有定義檔，提取出如下結構：

* **模組（Package）：** 對應於 ODL 中的 %config 區塊或檔案名稱，代表一個獨立的服務實體。
* **物件（Class）：** 對應於 ODL 中的 object 定義（例如 Device.WiFi.Radio）。這些將成為視覺化中的二級節點。
* **函數（Function）：** 對應於 ODL 中的函數定義（例如 reset(), scan()）。這些是 API 互動的端點。

透過這種靜態分析，我們可以預先建立一張「類別地圖（ClassMap）」，作為視覺化的底圖。這確保了即使某些模組在追蹤期間未活躍，其結構仍能被呈現，滿足使用者對「架構圖」完整性的期待。

### 2.2 動態行為捕捉：eBPF 與 ubus 的橋接

「模組間有連線」這一需求，直指 OpenWrt 的 IPC 機制。在 prplOS 中，跨模組通訊主要透過 libubus 函式庫進行。要視覺化這些連線，必須攔截並解析 ubus 的訊息流。

**eBPF 的觀測點選擇：**

利用 eBPF（特別是 bpftrace 或 libbpf）進行非侵入式追蹤是最佳選擇。關鍵的掛載點（Hook Points）包括：

1. ubus\_invoke 5： 這是發送同步請求的核心函數。透過掛載 uprobe 於此函數，我們可以捕獲呼叫者（Caller）的 PID、目標物件的 ID（Object ID）、呼叫的方法名稱（Method Name）以及參數內容（Message Blob）。
   * *資料轉換：* 在圖論中，這代表一條有向邊（Directed Edge）。源節點為 Caller PID 所屬的模組，目標節點為 Object ID 所對應的模組，邊的標籤即為方法名稱。
2. **ubus\_send\_event：** 用於廣播通知。這代表了一種「發布-訂閱（Pub-Sub）」關係，視覺化時可用不同顏色的虛線表示，以區別於直接的 API 呼叫。

**物件 ID 的解析難題：** ubus 在核心傳輸時使用整數 ID（例如 123）而非字串名稱（如 network.interface）。為了在圖表中顯示人類可讀的名稱，必須在追蹤開始時執行 ubus list 以建立 ID 與名稱的對照表，或者同時追蹤 ubus\_add\_object 事件以動態維護此對照表 7。

### 2.3 內部細節具象化：FSM 與 GDB 的整合

使用者特別強調「模組內含細節 (FSM/API)」。在 prplMesh 等複雜網通軟體中，FSM（有限狀態機）是控制邏輯的核心。例如，Beerocks Agent 使用 FSM 來管理 ASSOCIATED、AUTHENTICATING、OPERATIONAL 等狀態 8。

**FSM 的資料特徵：**

FSM 的狀態通常由 C/C++ 中的 enum 列舉型別表示，狀態變遷則由特定的變數賦值或巨集（Macro）觸發。例如，prplMesh 使用 FSM\_MOVE\_STATE 巨集來執行狀態切換並記錄日誌。

**GDB 與 eBPF 的互補應用：**

* **GDB 的角色：** 適用於開發階段的深度快照。利用 GDB 的 Python API 10，我們可以編寫腳本在 FSM\_MOVE\_STATE 處設置斷點，並提取當前的呼叫堆疊（Call Stack）與局部變數。這能提供極其詳細的上下文，但會暫停程式執行，不適合即時監控。
* **eBPF 的角色：** 適用於執行時期的流式追蹤。透過在狀態變遷函數上設置 uprobe，可以連續記錄狀態變化的時間序列（Time-series）。
* **視覺化映射：** 在圖形中，FSM 的每一個「狀態」可以被視為模組內部的一個子節點。狀態變遷事件則轉化為這些子節點之間的連線。或者，利用 AppMap 的概念，將「狀態」視為一段長執行的「函數」，從進入狀態到離開狀態的時間區間，即為該函數的執行時間。

## 3. 現有工具適用性評估

針對使用者詢問的 AppMap、Sourcetrail 與 Cytoscape.js，本節進行深入的適用性分析，比較其在 OpenWrt/prplOS 環境下的優劣。

### 3.1 AppMap：最佳的互動式架構瀏覽體驗

**評估結論：高度推薦，需開發資料轉接器（Adapter）。**

AppMap 是目前市場上最接近使用者描述之「展開式架構圖」的工具。它專為呈現程式碼結構（Structure）與執行行為（Behavior）而設計，其核心價值在於同時提供「依賴關係圖（Dependency Map）」與「時序圖（Sequence Diagram）」 12。

* **符合度分析：**
  + **展開式介面：** AppMap 的 UI 原生支援 Package -> Class -> Function 的層次結構導航。使用者可以點擊一個 Package（對應 OpenWrt 模組），展開查看其中的 Classes（對應 ODL 物件），再查看 Functions（對應 API 或 FSM 狀態）。這完全契合使用者的視覺需求。
  + **連線視覺化：** 它自動繪製模組間的呼叫連線，並允許使用者點擊連線以查看具體的參數與回傳值 13。
  + **序列圖整合：** 這是 AppMap 的殺手級功能。對於 ubus 這種非同步通訊，能夠以時序圖呈現訊息的流動（Message Flow），對於除錯 Race Condition 或理解複雜互動至關重要 14。
* **實作挑戰：**
  + AppMap 目前主要支援 Ruby, Java, Python, Node.js，缺乏官方的 C/C++ Agent。
  + **解決方案：** AppMap 的資料格式是公開且文檔齊全的 JSON 規範 15。我們可以編寫一個後處理腳本，將 GDB/eBPF 收集到的原始日誌「翻譯」為符合 AppMap 規範的 appmap.json 檔案。這是一種「偽裝」策略，讓 AppMap 誤以為它在顯示一個 Java 程式，但實際上顯示的是 C++ 的 OpenWrt 系統行為。

### 3.2 Cytoscape.js：最強大的客製化圖形引擎

**評估結論：技術可行，但開發成本極高。**

Cytoscape.js 是一個純 JavaScript 的圖論視覺化函式庫，而非開箱即用的應用程式 16。它提供了繪製複雜網路圖所需的底層原語（Primitives）。

* **符合度分析：**
  + **複合節點（Compound Nodes）：** Cytoscape.js 原生支援節點的巢狀結構（Nesting），這是實現「模組內含細節」的基礎技術 17。
  + **擴充性：** cytoscape-expand-collapse 擴充套件 18 提供了開箱即用的展開/收合互動邏輯，完全符合使用者手繪圖的操作模式。
  + **佈局演算法：** CoSE-Bilkent 等佈局演算法專為複合圖設計，能自動處理節點展開後的排版問題，避免圖形重疊或混亂 20。
* **實作挑戰：**
  + 使用者必須自行開發網頁前端（HTML/JS/CSS）來託管圖表。
  + 需要自行實作資料載入、過濾、搜尋以及與 GDB 數據的整合邏輯。它像是一盒樂高積木，能堆出任何形狀，但你需要自己動手堆。

### 3.3 Sourcetrail：靜態導航的王者，動態視覺化的弱者

**評估結論：不推薦用於此特定需求。**

Sourcetrail 是一款優秀的靜態原始碼瀏覽工具 21，擅長分析「誰呼叫了誰」的靜態關係。

* **符合度分析：**
  + **靜態 vs 動態：** Sourcetrail 的核心資料庫建立在編譯時期的符號索引上。它無法展現「執行時期」的動態行為，例如「在 T 時間點，模組 A 透過 ubus 發送了訊息給 B」。對於依賴 IPC 的 OpenWrt 架構，靜態分析無法看見跨行程的關聯。
  + **FSM 限制：** 雖然可以透過 SDK 注入自定義數據 22，但 Sourcetrail 的介面不具備「時序播放」或「狀態變遷」的視覺語意。它展示的是程式碼的地圖，而非程式執行的軌跡。
  + **維護狀態：** 該專案已停止維護 23，對於需要長期支援的專案而言風險較高。

## 4. 實作方案：「prplVis」資料轉換管線設計

基於上述分析，本報告提出一套基於 **AppMap** 的實作方案，因為它在滿足「可展開」與「連線」視覺化需求的同時，開發成本遠低於從頭構建 Cytoscape 應用。

本方案的核心在於構建一個「轉換器（Transformer）」，將 OpenWrt 的異質數據標準化為 AppMap JSON 格式。

### 4.1 階段一：資料獲取 (Data Acquisition)

我們需要從兩個維度收集數據：靜態結構與動態行為。

1. **靜態結構提取（Static Extraction）：**
   * 利用 amxo-cg 或 libamxo 的 Python 綁定 1 解析系統中的 .odl 檔案。
   * 解析目標：提取 object（物件名）、function（方法名）以及 %config 中的模組標識。這些數據將構成 AppMap 中的 classMap（類別地圖）。
2. **動態行為錄製（Dynamic Recording）：**
   * **IPC 追蹤：** 使用 bpftrace 腳本掛載於 libubus.so。  
     C  
     /\* bpftrace 偽代碼示例 \*/  
     uprobe:/lib/libubus.so:ubus\_invoke {  
      $ctx = arg0; $obj = arg1; $method = str(arg2);  
      /\* 記錄時間戳、呼叫者PID、目標物件ID、方法名 \*/  
      printf("EVENT\_CALL|%d|%d|%d|%s\n", nsecs, pid, $obj, $method);  
     }
   * **FSM 狀態追蹤：** 針對 prplMesh，追蹤 FSM\_MOVE\_STATE 相關的函數。若原始碼中有統一的狀態切換函數（如 set\_state），則直接掛載；若為巨集，則需追蹤巨集展開後寫入狀態變數的指令位址，或在編譯時插入 Tracepoints（SDT）。

### 4.2 階段二：資料正規化與轉換 (Normalization & Transformation)

編寫一個 Python 腳本（prpl\_to\_appmap.py）處理上述數據。

**類別地圖（ClassMap）建構：**

AppMap 的 classMap 需要 package、class、function 的層次。映射邏輯如下表：

| **AppMap 概念** | **OpenWrt/prplOS 對應實體** | **範例** |
| --- | --- | --- |
| **Package** | 系統行程或守護行程 (Daemon) | netifd, beerocks\_agent |
| **Class** | Ambiorix 資料模型物件 (Data Object) | Device.WiFi.Radio, BackhaulManager |
| **Function** | ubus 方法或 FSM 狀態 | scan, State: CONNECTING |

**事件序列（Events）建構：**

AppMap 的 events 陣列描述執行流程。

* 將 ubus\_invoke 的開始映射為 call 事件，ubus 請求完成映射為 return 事件。
* **關鍵創新：** 將 FSM 的狀態視為「函數」。當 FSM 進入 CONNECTING 狀態時，生成一個 call function "State: CONNECTING" 的事件；當離開該狀態時，生成對應的 return 事件。這樣，AppMap 的火焰圖（Flame Graph）將直觀地顯示系統在各個狀態停留的時間，這對於效能分析與死鎖偵測極具價值。

### 4.3 階段三：視覺化呈現 (Visualization)

生成的 trace.appmap.json 檔案可直接載入 VS Code 的 AppMap 擴充套件中。

* **相依性圖視圖：** 使用者將看到 beerocks\_agent 節點指向 hostapd 節點。點擊 beerocks\_agent，節點展開，顯示內部的 BackhaulManager 物件及 State: OPERATIONAL 等狀態節點。
* **時序圖視圖：** 使用者可以查看特定時間段內，FSM 狀態切換與外部 ubus 呼叫的先後順序。例如，清楚呈現「進入 ASSOCIATING 狀態 -> 發送 ubus call hostapd associate -> 收到回覆 -> 狀態切換為 ASSOCIATED」的完整流程。

## 5. 深入技術探討與實作細節

### 5.1 處理 ubus 的非同步特性

ubus 支援非同步呼叫（Async Calls），這意味著請求與回應可能在時間上相距甚遠，且可能穿插其他事件。在轉換為 AppMap 格式時，必須利用 ubus\_request 結構中的 seq（序號）來正確配對 call 與 return 事件 5。若 eBPF 漏失了回應事件，轉換腳本需具備逾時補償機制，避免圖表中的呼叫堆疊無限增長。

### 5.2 FSM 狀態的符號解析

GDB/eBPF 抓取到的狀態值通常是整數（Enum Value）。為了讓視覺化具備可讀性，轉換腳本需要讀取原始碼中的 enum 定義或 GDB 的符號表，將 State: 3 自動轉換為 State: CONNECTING。對於 prplMesh，這可以通過解析其 C++ 標頭檔或利用 GDB 的 gdb.lookup\_type API 來自動完成 25。

### 5.3 效能與開銷考量

在嵌入式設備上運行 eBPF 雖然效能損耗較低，但大量的 ubus 流量仍可能產生巨量日誌。建議在 bpftrace 腳本中引入過濾機制（Filter），例如僅追蹤特定 PID 或特定 ubus 路徑（如 Device.WiFi.\*），以確保即時性並減少雜訊 27。

## 結論

總結來說，利用 GDB/eBPF 數據在 OpenWrt/prplOS 環境下實現「可展開且具備連線」的軟體架構視覺化不僅是可行的，而且是提升該領域開發效率的強大手段。

雖然缺乏現成的 C/C++ 專用工具，但透過\*\*資料轉換（Data Transformation）\*\*的方法，我們可以有效地利用 **AppMap** 現有的強大前端介面。這種方法避免了從零開發視覺化引擎的成本，同時能夠完美呈現 FSM 狀態變遷與 IPC 通訊的複雜互動。對於有高度客製化需求的情境（例如整合至路由器 Web 管理介面），則建議基於 **Cytoscape.js** 進行二次開發。

建議開發團隊優先採用 **AppMap JSON 轉接器** 方案作為概念驗證（PoC），這將能以最低的成本驗證視覺化帶來的除錯與架構分析價值。

*(本報告內容係基於對 OpenWrt/prplOS 架構特性及各視覺化工具技術規範的綜合分析。)*

#### 引用的著作

1. doc/odl.md · main · prpl Foundation / components / ambiorix / libraries / libamxo - GitLab, 檢索日期：2月 7, 2026， <https://gitlab.com/prpl-foundation/components/ambiorix/libraries/libamxo/-/blob/main/doc/odl.md>
2. prpl Foundation / components / ambiorix / tutorials / getting-started - GitLab, 檢索日期：2月 7, 2026， <https://gitlab.com/prpl-foundation/components/ambiorix/tutorials/getting-started>
3. prpl Foundation / components / ambiorix / tutorials / datamodels / introduction - GitLab, 檢索日期：2月 7, 2026， <https://gitlab.com/prpl-foundation/components/ambiorix/tutorials/datamodels/introduction/-/tree/main>
4. JohnHedman/odl-parser - GitHub, 檢索日期：2月 7, 2026， <https://github.com/JohnHedman/odl-parser>
5. OpenWRT UBUS - HackMD, 檢索日期：2月 7, 2026， <https://hackmd.io/@rYMqzC-9Rxy0Isn3zClURg/H1BY98bRw>
6. ubus/libubus.h at master · txomon/ubus - GitHub, 檢索日期：2月 7, 2026， <https://github.com/txomon/ubus/blob/master/libubus.h>
7. [OpenWrt Wiki] ubus (OpenWrt micro bus architecture), 檢索日期：2月 7, 2026， <https://openwrt.org/docs/techref/ubus>
8. prplMesh/agent/src/beerocks/slave/backhaul\_manager/backhaul\_manager\_thread.cpp at master - GitHub, 檢索日期：2月 7, 2026， <https://github.com/prplfoundation/prplMesh/blob/master/agent/src/beerocks/slave/backhaul_manager/backhaul_manager_thread.cpp>
9. prplMesh - GitLab, 檢索日期：2月 7, 2026， <https://gitlab.com/prpl-foundation/prplmesh/prplMesh/-/tree/master>
10. gdbBacktraceToJson.py parses the output of the command "thread apply all bt full" and turns it into a json array. Useful for automating the analysis of coredump files generated when an application crashes. Use the tokenised json to search a database of crashes for similar crashes or make a nice web interface for viewing back traces. You could make a backtrace diff tool. It's much easier to use and write tools using a standard data format. - GitHub Gist, 檢索日期：2月 7, 2026， <https://gist.github.com/JodiTheTigger/8411686>
11. Frames In Python (Debugging with GDB) - Sourceware, 檢索日期：2月 7, 2026， <https://sourceware.org/gdb/current/onlinedocs/gdb.html/Frames-In-Python.html>
12. Docs - Guides - Using AppMap Diagrams, 檢索日期：2月 7, 2026， <https://appmap.io/docs/reference/guides/using-appmap-diagrams.html>
13. Visualization in your code editor - AppMap, 檢索日期：2月 7, 2026， <https://appmap.io/product/appmap-in-the-code-editor.html>
14. Quickly learn how new-to-you code works using sequence diagrams - AppMap, 檢索日期：2月 7, 2026， <https://appmap.io/blog/2023/06/06/quickly-learn-how-new-to-you-code-works-using-sequence-diagrams/>
15. Specification for AppMap clients - GitHub, 檢索日期：2月 7, 2026， <https://github.com/getappmap/appmap>
16. Cytoscape.js, 檢索日期：2月 7, 2026， <https://js.cytoscape.org/>
17. Compound Nodes in Cytoscape JS - javascript - Stack Overflow, 檢索日期：2月 7, 2026， <https://stackoverflow.com/questions/17460987/compound-nodes-in-cytoscape-js>
18. @bardit/cytoscape-expand-collapse - NPM, 檢索日期：2月 7, 2026， <https://www.npmjs.com/package/@bardit/cytoscape-expand-collapse>
19. iVis-at-Bilkent/cytoscape.js-expand-collapse - GitHub, 檢索日期：2月 7, 2026， <https://github.com/iVis-at-Bilkent/cytoscape.js-expand-collapse>
20. PDF version - Cytoscape User Manual, 檢索日期：2月 7, 2026， <https://cytoscape.org/manual/Cytoscape3_9_0Manual.pdf>
21. Use Sourcetrail to Work Easily with Unfamiliar Code - Open Source For You, 檢索日期：2月 7, 2026， <https://www.opensourceforu.com/2020/10/use-sourcetrail-to-work-easily-with-unfamiliar-code/>
22. Leveraging Sourcetrail to a mapping tool, meet Numbat and Pyrrha - Quarkslab's blog, 檢索日期：2月 7, 2026， <https://blog.quarkslab.com/leveraging-sourcetrail-to-a-mapping-tool-meet-numbat-and-pyrrha.html>
23. CoatiSoftware/Sourcetrail: Sourcetrail - free and open-source interactive source explorer - GitHub, 檢索日期：2月 7, 2026， <https://github.com/CoatiSoftware/Sourcetrail>
24. README.md · v1.0.4 · prpl Foundation / components / ambiorix / applications / amxo-cg, 檢索日期：2月 7, 2026， <https://gitlab.com/prpl-foundation/components/ambiorix/applications/amxo-cg/-/blob/v1.0.4/README.md>
25. Debugging in GDB: Create custom stack winders - Red Hat Developer, 檢索日期：2月 7, 2026， <https://developers.redhat.com/articles/2023/06/19/debugging-gdb-create-custom-stack-winders>
26. gdb-python : Parsing structure's each field and print them with proper value, if exists, 檢索日期：2月 7, 2026， <https://stackoverflow.com/questions/16787289/gdb-python-parsing-structures-each-field-and-print-them-with-proper-value-if>
27. bpftrace/bpftrace: High-level tracing language for Linux - GitHub, 檢索日期：2月 7, 2026， <https://github.com/bpftrace/bpftrace>