Lehr- und Forschungseinheit
SW-Technik

23. Nowember 1998

Die Programmierumgebung SNiFF+

- eine kurze Anleitung -

Inhalt

0. Vorbemerkung

Diese Schrift ist vorrangig für das Praktikum zur PI3-Vorlesung von Prof. Bothe gedacht.

Damit
• erfolgt eine inhaltliche Beschränkung auf die Sprachen C/C++,
• werden nur die wichtigsten, aber ausreichenden Leistungen von SNiFF beschrieben,
• werden praktische Hinweise zur Einbettung der Übungsaufgaben in SNiFF angeboten.

Die Schrift ist eine Ergänzung zu den Ausführungen in den Praktikumsveranstaltungen und zu der SNiFF-Demonstration.

Bezugspunkt ist **SNiFF Release 3.1**

1. Überblick

SNiFF ist an unserem Institut auf den Sun-Rechnern unter Solaris verfügbar.

SNiFF besitzt ein breites Leistungsspektrum, von dem Sie insbesondere drei Leistungen nutzen werden:
a) Es schafft Ordnung, indem es Sie bei der Organisation und Verwaltung Ihrer Übungsaufgaben und zugehörigen Sourcefiles unterstützt (siehe weiter unten).
b) Es bietet Ihnen einen übersichtlichen und effizienten Edit-Compile-Link-Debug-Zyklus (siehe 2.2.).
c) Schließlich erhalten Sie zahlreiche Informationen über Strings und Symbole (Funktionen, Variablen, ...) und über include-Hierarchien aus Ihren Sourcecode-Dateien (Implementations- und Header-Dateien) (siehe 2.3.).

SNiFF bietet noch mehr, was jedoch auf Grund der einschränkenden „Vorbemerkung“ nicht weiter betrachtet wird. So z.B.
• Hilfen bei der Erstellung der Programmdokumentation
• Generierung von make-Files (wir benutzen eigene Makefiles)
• Versions- und Konfigurationsmanagement
• Unterstützung bei der Softwareentwicklung im Team

Eine ausführliche Dokumentation ist online unter /vol/tempus-vol2/SNiFF+2.4/doc verfügbar, hier in gepacktem ps-Format (entpacken mit gunzip). Besonders verwiesen sei auf:
- CPP_FortranTutorial.ps.Z
- ReferenceGuide.ps.Z
- UsersGuide.ps.Z

Des Weiteren bietet SNiFF im Rahmen seiner HELP-Funktion eine umfangreiche Dokumentation an.

Bevor in den Folgepunkten auf spezielle SNiFF-Tools eingegangen wird, hier noch einige allgemeine Bemerkungen:
(Anmerkung zum verwendeten Tool-Begriff: Die Autoren von SNiFF verwenden den Begriff sowohl für das Werkzeug insgesamt, als auch für jedes Teilwerkzeug innerhalb von SNiFF.)

Jedes SNiFF-Tool hat sein eigenes Fenster mit eigener Funktionsvielfalt, die vorrangig über Pull-down-Menüs und ergänzend über Pop-up-Menüs angeboten wird. Das erste Menü innerhalb der Menüleiste ist immer das Ikonmenü. Es ist in seiner Form toolspezifisch und bietet nach dem Auswählen zusätzlich alle übrigen Tools an.

Zustandsabhängig kann das toolspezifische Ikon durch zwei andere ersetzt sein:

Objekt ist read only (z.B. Textfile)

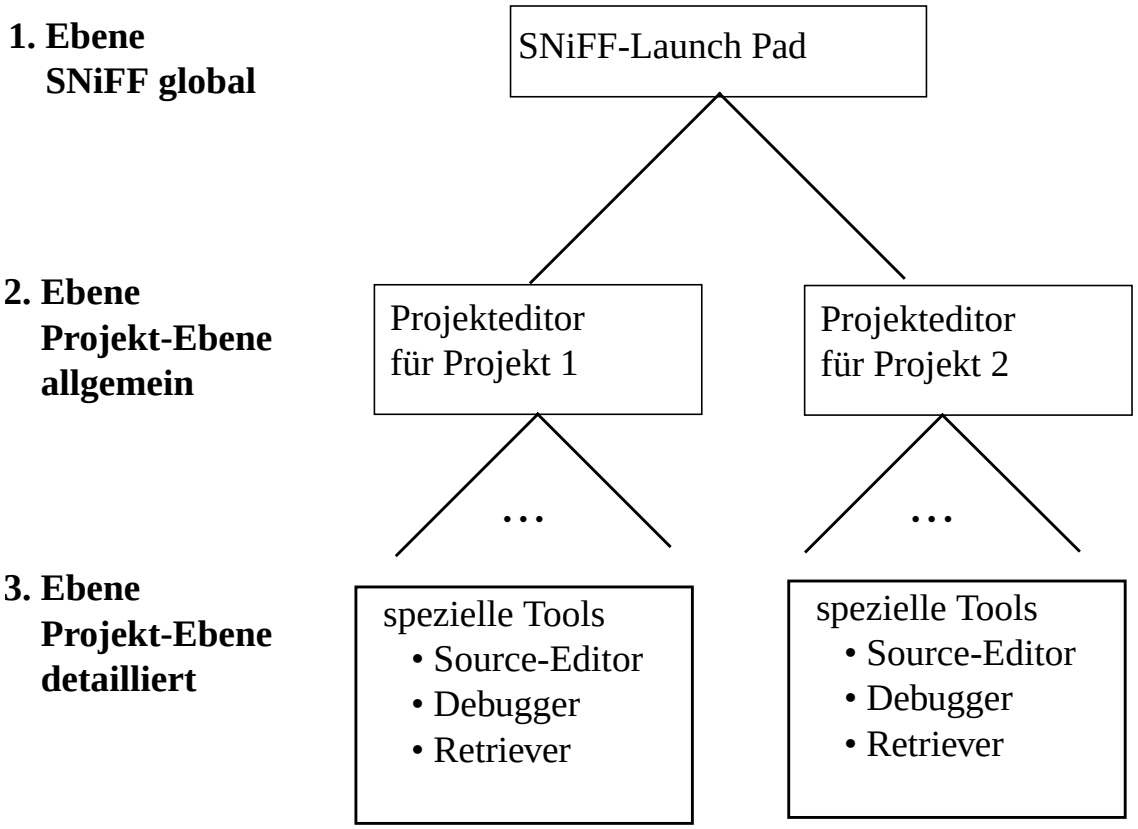
Objekt modifiziert und noch nicht gesichert. (Hierrauf sollten Sie besonders achten und zur Wiederherstellung von Konsistenz die Save-Funktion nutzen.)

Insgesamt werden Ihnen folgende Tools bereitgestellt:

- Project Editor
- Symbol Browser
- Editor
- Class Browser
- Retriever

- Hierarchy Browser
- Cross Referencer
- Include Browser
- Configuration Manager
- Diff Merger
- Documentation Editor
- Shell

Die SNiFF-Tools bilden eine 3-Ebenen-Architektur:



zum Launch Pad siehe 3.2. und 3.3
zum Projekteditor siehe 3.3.
zu den speziellen Tools siehe 2.2. und 2.3.

PS: Übrigens heißt ‘sniff‘ schnüffeln und bezieht sich auf ein schnüffelndes Schwein, - daher auch die zahlreichen Schweineikone. Möge das Schwein in Ihren zahlreichen Dateien, Symbolen und ‘Programmwanzen‘ erfolgreich herumschnüffeln.

2. Tools und die zwei Symbolwelten

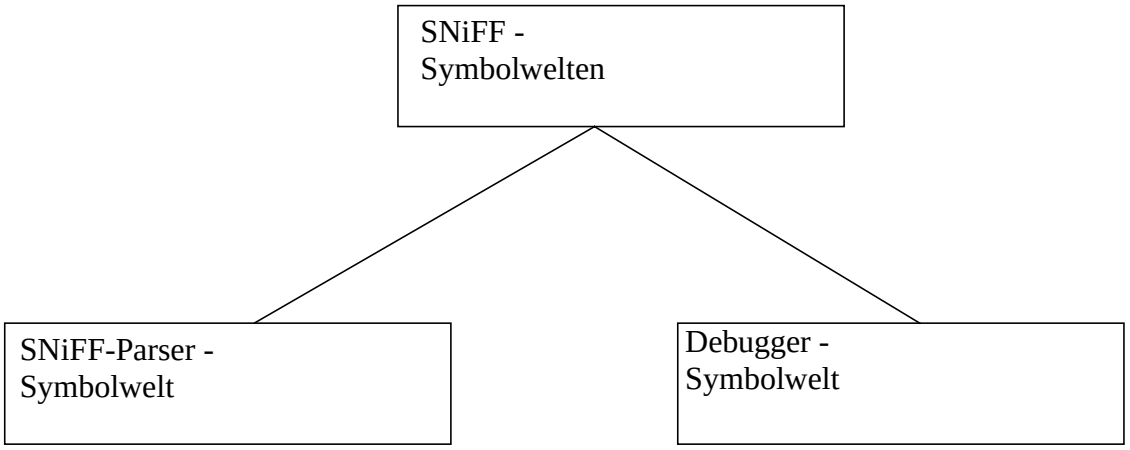
2.1 Einführung

Bei der Arbeit mit SNiFF spielen Angaben zu den in Ihren Sourcefiles verwendeten Symbolen eine zentrale Rolle. Sie bilden die Basis vieler SNiFF-Tools.

Diese Symboldaten werden nun an zwei unterschiedlichen Stellen geführt, von unterschiedlichen Tools

erzeugt, verwaltet und verwendet, - sie bilden zwei Welten.

Es ist sehr hilfreich, diese zweigeteilte Sicht ständig präsent zu haben.



• Erzeugung/Modifikation
  Durch den SNiFF-Parser (siehe 2.3.).
  Dieser wird intern u.a. aufgerufen bei:
  - Übernahme/Entfernen von Source-files in/aus ein(em) Projekt
  - Sourcecode-Editor-Abschluß oder bei zwischenzeitlichem ‘save‘
  - Übernahme/Entfernen eines Sub-projektes in/aus ein(em) Projekt

• Inhalt
  siehe 2.3.

• Ort
  Symbolspeicher-File in ‘.sniffdir‘
  (siehe 3.3.)

• Nutzung:
  u.a. durch den Browser, Retriever und den Cross-Referencer (siehe 2.3.)

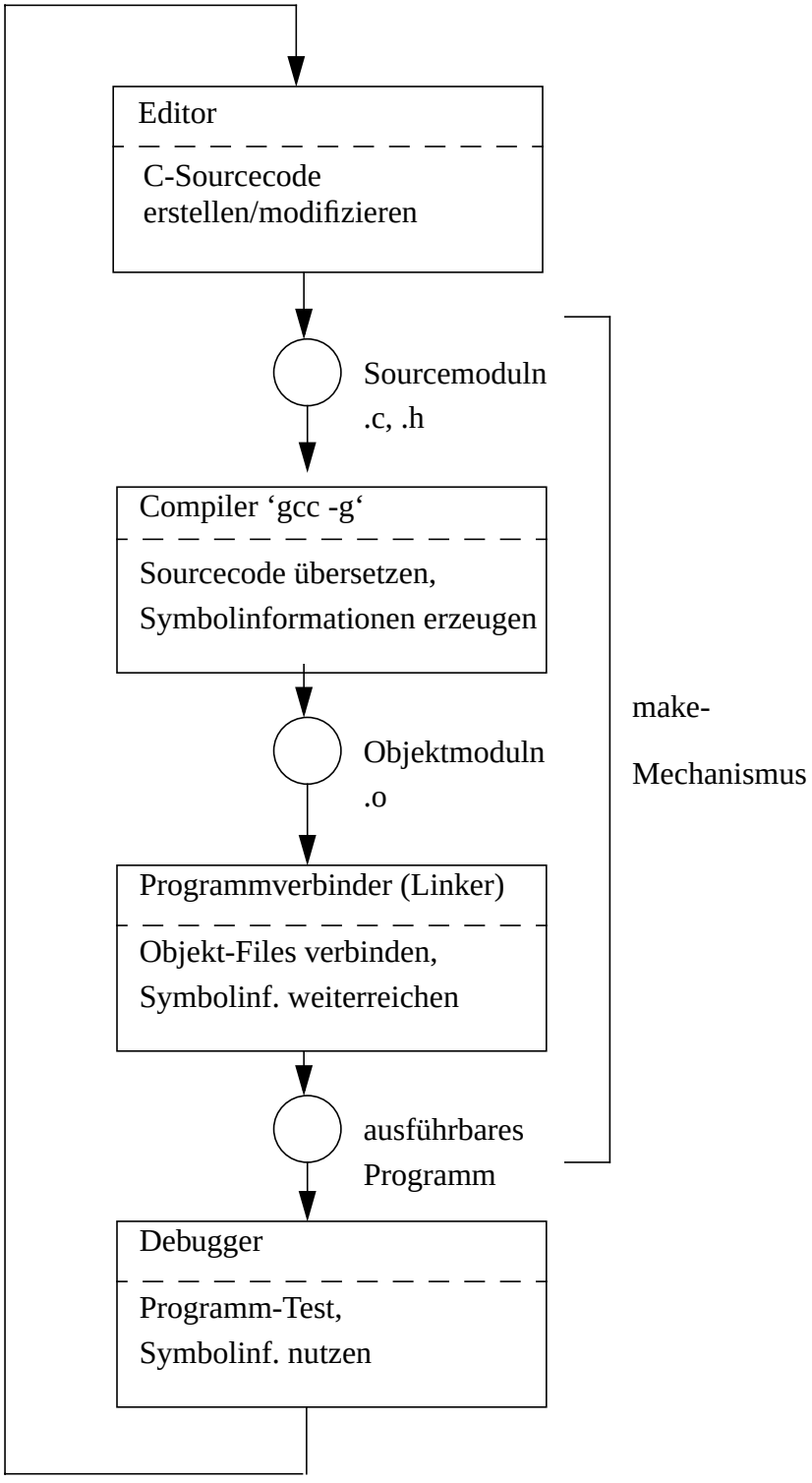
• Erzeugung
  Durch den Compiler bei Parameterangabe ‘-g‘.
  (Werden durch den Programmverbinder weitergereicht.)

• Inhalt
  - Für jede Variable:
    Name, Datentyp und Adresse
  - Für jede Funktion:
    -- Name,Datentyp und Adresse
    -- Name, Datentyp und Adresse im Stackrahmen von allen Parametern und lokalen Variablen.
  - Für jede Sourcecode-Zeile:
    Anfangsadr. der Maschinenbefehle, die die Übersetzung dieser Zeile darstellen.
  (Damit mehr, als nur Symboldaten.)

• Ort
  Teile der Objektmoduln und des ausführbaren Programms

• Nutzung
  Beim Programmtest durch den Debugger (siehe 2.2.)

2.2. Programmierzyklus Edit-Compile-Link-Debug



Editor (Source Editor)

Der Editor (hier Source-Editor im Unterschied zum Dokumenten-Editor) wird entweder über das Ikon-Menü gestartet, oder kontextsensitiv, indem Sie bspw. im Projekteditor auf ein Sourcefile klicken, im Symbolbrowser auf ein Symbol oder im Retriever auf ein String. Immer wird ein Editorfenster geöffnet, das entsprechende File geladen und auf das Symbol oder String positioniert.

Der Editor strukturiert Ihr Sourcefile, indem er Symbole und Kommentare mit unterschiedlichen Fonts und Farben darstellt.

Die Editorfunktionen werden Ihnen unter den Menüs ‘Edit’ und ‘Positioning’ angeboten. Von ihnen sind Ihnen viele bekannt, teilweise sind sie selbsterklärend, oder sie lassen sich durch Ausprobieren schnell deuten, - auf sie wird hier nicht eingegangen. Zu einigen Funktionen folgende Erklärungen:

File	-Revert	Zurückstellen auf die letzte Sicherung
	-Save	zwischenzeitliches Sichern (mit Parser-Aufruf und Aktualisierung der SNiFF-Symboltabelle, **vorausgesetzt, Ihr Sourcefile ist Bestandteil des Projektes (siehe 3.3.)**)
Positioning	-Find/Change	Auffinden und wahlweises Ersetzen von Strings
	-Edit Header-/Impl. File	Wechsel zum entsprechenden File
	-Go to Line	Sprung zu Zeilennummer; eingangs wird immer die Zeilennummer der aktuellen Kursorposition angezeigt.

Compiler, Linker

Verwendet wird der GNU-C-Compiler. Zu ihm und dem Linker siehe 3.3., 3.4.

Make-Mechanismus

Siehe 3.3 und 3.4

Debugger

Benutzt wird der Debugger gdb, ein nichtgraphischer, kommandoorientierter Sourcecode-Debugger. SNiFF stellt für ihn ein graphisches und menüorientiertes Front-End bereit, wodurch die Arbeit mit ihm einfach und sehr anschaulich wird.

Der Start des Debuggers erfolgt z.B. über das Editorfenster: Target - Debug. Im anschließend geöffneten Debuggerfenster werden über Menüleiste zahlreiche Funktionen angeboten. Die wichtigsten von Ihnen erscheinen nach dem Debugger-Aufruf zusätzlich im Editorfenster als Button -Leiste. Weitere Funktionen können darüberhinaus hinter dem Debugger-Prompt ‘gdb>‘ in Kommandoform angefordert werden. Eine Kommandoübersicht bietet ‘help‘.

Neben der Kommandoeingabe dient das Debuggerfenster während der Programmausführungszeit auch als Ein-/Ausgabemedium für stdin, stdout und stderr.

Aus der Funktionsvielfalt werden nun drei Funktionskomplexe beschrieben:

a) Unterbrechungspunkte (UP) setzen/anzeigen/löschen
An einem UP stoppt die Programmausführung und es werden Analyse-, Modifikations- und Fortsetzungsmöglichkeiten geboten.

• UP an beliebiger Stelle
Setzen UP: Kursor auf gewünschte Sourcecode-Zeile; Funktion BreakAt (Anzeige UP-Symbol).
Anzeige aller UP: Funktion Breakpoints-Browse.
LöschenUP: Kursor in UP-Sourcecode-Zeile und Funktion Clear oder über Breakpoints - Browse -

Clear.
• UP am Funktionsanfang
Setzen: Mit Kursor Funktionsnamen auswählen (Doppelklick) und Funktion BreakIn.
Anzeige/Löschen: siehe oben.

b) Abarbeitung des zu testenden Programmes
• Programmstart aus Editorfenster:
Run-Button; anschließend werden wahlweise zu übergebende Argumente abgefragt (z.B. ‘Arguments: -T <bsp.pas’ für das Argument ‘-T’ und Standardeingabe von der Datei ‘bsp.pas’.
• Programmstart aus Debuggerfenster:
Execution - Run; anschließend keine Möglichkeit der Parameterübergabe. Dies ist aus dem Debuggerfenster heraus nur über das run-Kommando möglich. Desgleichen muß das run-Kommando benutzt werden, wenn man Ein-/Ausgabeumlenkung haben möchte.
run-Syntax: run [param] [<infile] [>|>>outfile]

Um die Debuggerfunktionen anschließend anwenden zu können, benötigen Sie Unterbrechungspunkte. Diese setzen Sie entweder vor dem Programmstart (mindestens einen), oder Sie unterbrechen das gestartete Programm mit der Funktion Execution - Interrupt. Nun können Sie Debuggerfunktionen ausführen und das Programm mit den Funktionen Cont, Next oder Step fortführen (siehe unten). Die aktuelle Abarbeitungsposition wird immer parallel in Ihrem Sourcecode-File angezeigt.

• Abarbeitungsfortsetzung:

Cont	Abarbeitungsfortsetzung bis zum nächsten UP; Fortsetzung in diesem Modus mit der Return-Taste.
Step	Anweisungsweise Abarbeitung einschließlich aufgerufener Funktionen. (Wirkt wie ein UP nach der nächsten Anweisung.) Fortsetzung in diesem Modus mit der Return-Taste.
Next	Wirkt wie ‘Step‘, nur werden aufgerufene Funktionen übergangen.

c) Anzeigen, Ändern und Lokalisieren von Daten
• Anzeigen: Kursor auf gewünschten Bezeichner positionieren (z.B. Variablenname). Nachfolgend:

Print / Print*	Einmalige Anzeige des entsprechenden Wertes im Dialogfenster.
Debug: Display / Display*	Fortlaufende Anzeige des entsprechenden Wertes im gesonderten Dialogfenster.
Anzeige inaktivieren:	z.B. über Display-Fenster: Tool - Clear

Vielfältig sind auch die Möglichkeiten mittels des gdb-Kommandos ‘info‘.
• Ändern: Hier nur der Verweis auf die entsprechenden gdb-Kommandos (Anzeige über ‘help‘).
• Lokalisieren: Geeignet sind die Info-Funktionen (siehe 2.3.).

2.3. Browser, Retriever, CrossReferencer und SNiFF-Parser

SNiFF-Parser
Dieser Parser ist leistungsfähig und fehlertolerant. Seine Kurzcharakteristik erfolgte in 2.1. Die durch ihn aus dem Sourcecode herausgefilterten und in der SNiFF-Symboltabelle abgelegten Symbole sind folgende:

Type indicator	Symbol type	Type indicator	Symbol type
cl	class	en	Enumeration
cd	constructor or destructor	ei	Enumeration item
iv	Instance variable	td	Typedef

me	Method	te	Template
f	Function	st	Structure
v	Global variable	un	Union
co	Constant variable	pc	Primitive C data type (char, ...)
ma	Macro	ud	Undefined symbol (z.B. die Funktion printf ist nicht im Projekt def.)

Den entsprechenden Typindikator verwendet bspw. der CrossReferencer. Er bietet auch noch weitere Informationen aus der Symboltabelle an:

Access indicator	Access mode
r	Read
w	Write
H	Used in as component type (has-a relationship)
P	Used as a parameter type
R	Used as return value type

Browser, Retriever, CrossReferencer

Diese Tools dienen einer weitreichenden Navigation und Übersichtsdarstellung auf der Basis von Symbolen oder beliebigen Strings. Dabei kann das Bezugsobjekt das aktuelle File, das Projekt oder es können alle Projekte sein.

Tool - Übersichtstabelle:

Tool	Basis	Bezeichnung im Info(I)- oder Class(C)-Menü
Symbol Browser	Symbol (SNiFF-Symbol*)	I: Find Symbols ...
Class Browser (oo)	Symbol (SNiFF-Symbol*)	C: Browse
Hierarchy Browser (oo)	Symbol (SNiFF-Symbol*)	C: Show in ...
Cross Referencer	Symbol (SNiFF-Symbol*)	I: Refers-To/Referred-By
Retriever	String	I: Retrieve ...

* SNiFF-Symbol aus der SNiFF-Parser-Symbolwelt.

Bei der Arbeit mit diesen Tools sind immer die Sourcecode-Files der zentrale Bezugspunkt. D.h., wenn Sie innerhalb eines Toolfensters ein Symbol oder einen String auswählen (Doppelklick), wird automatisch die entsprechende Stelle im zugehörigen Sourcecode-File angezeigt. Umgekehrt können Sie ein Symbol oder einen String im Sourcecode auswählen und gelangen anschließend über die Funktionen Info-Retrieve, -Find oder -Reference in die entsprechenden Tools.

Symbolbrowser

Der Symbolbrowser bietet einen Überblick über die im Sourcecode definierten externen Symbole. Dabei kann die Auswahl über drei unterschiedliche Angaben im Symbolbrowser-Fenster spezifiziert werden:
- Type pop-up-Menü (links): Gestattet die Auswahl anzuzeigender Symboltypen oder zeigt den Typ ausgewählter Symbole an.
- Filter-Feld: Angabe des Symbolnamens. Für eine verallgemeinerte Angabe sind als Muster GNU-regu-

läre Ausdrücke erlaubt (Ausdruckssyntax wie beim emacs-Editor).

• Projektbaum: Auswahl der in den Suchprozeß einzubeziehenden Projekte (bei Projekt-Subprojekt-Architektur).
Signature-Buttom: Anzeige zusätzlicher Angaben zu den Symbolen.

Retriever

Der Retriever ermöglicht die Suche nach beliebigen Strings. Bei erfolgreicher Suche werden alle Sourcecode-Zeilen, die das String enthalten, einschließlich der dazugehörigen Sourcefile-Namen, angezeigt. Die Auswahl kann über drei unterschiedliche Angaben im Retriever-Fenster spezifiziert werden:
• String-Feld: Eingabe des Strings
• Filter-Feld:
  a) Eingabe eines Strings oder eines als Muster dienenden regulären Ausdruckes (siehe Filter zum Symbolbrowser). Wirkung: Die String-Feld-Angabe wird einem zweiten Filterprozeß unterzogen.
  b) Man kann auch über das Filtermenü vordefinierte Filter wählen (z.B. assigment od. comparison).
• Projektbaum: siehe unter Symbolbrowser.

Cross Referencer

Ausgehend von einem ausgewähltenSymbol bietet der Cross Referencer graphisch aufbereitet:
• Bezüge zu anderen Symbolen (Refers-To) oder
• Bezüge zu dem ausgewählten Symbol (Referred-By).

Als graphische Form wird die Baum- oder die eingerückte Form angeboten (Graph - Layout- ...). Die Verweistiefe ist wählbar (Symbol auswählen; anschließend Xref - Filter - Depht). Die Symbole werden mit ihren Symboltypen angezeigt (siehe ‘SNiFF-Parser‘ weiter oben).

Ergänzend zur graphischen Darstellung bietet das Cross Referencer-Fenster zur schnelleren Navigation ein Symbol-Index-Feld und zur Einschränkung der einbezogenen Symbolmenge ein Projektbaum-Feld an (siehe ‘Symbolbrowser‘ weiter oben).

Class Browser, Hierarchy Browser

Diese Tools bieten das Browsen über Methoden von Klassen und über Klassenstrukturen (Vererbung) an.

3. Die Arbeit mit SNiFF
3.1. Überblick

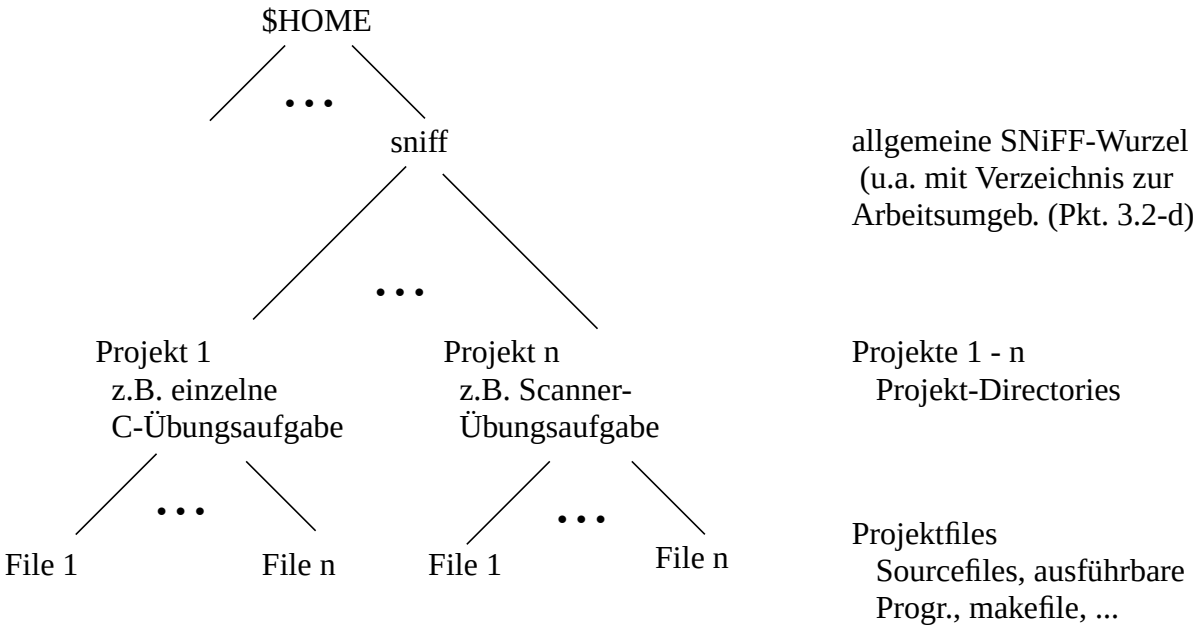
SNiFF bietet Ihnen hinsichtlich der Gestaltung Ihrer Programmierumgebung viele Möglichkeiten, so z.B.:
• Teilung Ihrer Projekte in Basis- und Subprojekte,
• Teilung der Entwicklungsumgebung zwischen mehreren Entwicklern (shared project),
• Unterstützung innerhalb des make-Mechanismus.
Hier soll nur ein einfacher und schnell praktizierbarer Umgang mit SNiFF beschrieben werden.

Das Einrichten Ihrer persönlichen SNiFF-Umgebung erfolgt in zwei Abschnitten:

a) Einmalige Aktivität (erster Abschnitt):
Einrichten eines SNiFF-Knotens unter Ihrem Home-Directory, einer Arbeitsumgebung (diese ist für das reine Browsen nicht erforderlich, siehe 3.2-d) und die Übernahme der Startshell (siehe 3.2.).

b) Mehrmalige Aktivitäten
Für jede Übungsaufgabe (*) definieren Sie ein SNiFF-Projekt und ordnen dieses als eigenes Directory unter Ihrem SNiFF-Knoten an (siehe 3.3.).
(*) Beziehen sich mehrere Übungsaufgaben auf die gleichen Sourcefiles und die Aufgaben gehören inhaltlich zusammen, können Sie diese zu einem Projekt vereinigen. Dies sollten Sie beim Praktikumscompiler bei den Aufgaben zum Scanner, zum Parser, zur Semantischen Analyse und zur Codegenerierung machen.

Empfohlene SNiFF-Directory-Hierarchie:



3.2. Umgebung, SNiFF-Präferenzen, SNiFF-Start (einmalige Aktivitäten)

a) Aktivitäten vor dem ersten SNiFF-Start

Voraussetzung ist, daß Sie sich auf einem Rechner mit dem Betriebssystem Solaris befinden. (Kontrolle: das Kommando ‘uname -a’ erzeugt die Ausgabe: SunOS <rechnername> **5.x** ...

- Anlegen der SNiFF-Wurzel
  %cd #Homedirectory
  %mkdir sniff # empfohlener Name
  %cd sniff
- Kopieren der SNiFF-Startshell in Ihr Verzeichnis ‘sniff’
  %cp ~sacklows/sniff/sniff+3.1.solaris.csh ./

b) Start von SNiFF

%cd sniff
%sniff+2.4.solaris.csh

Auswirkungen:
- Setzen der Umgebungsvariablen für das SNiFF-Installationsverzeichnis und für die Lizenzdatei
- SNiFF als Hintergrundprozeß starten
- Anlegen zweier Dateien in Ihrem Homedirectory (wenn bereits vorhanden, bleiben sie erhalten)
  - .UserPrefs.sniff SNiFF-User-Präferenzen (siehe c))
  - sniff.log SNiFF-Logdatei
- Eröffnen des 'Begin'-Fensters. Hier können Sie sich informieren, ob die SNiFF-Umgebung korrekt ist. Abschluß mit Finish-Button.
- Eröffnen eines Log-Fensters (von hier automatische Übernahme in die SNiFF-Logdatei 'sniff.log')
- Eröffnen des SNiFF-Startfensters 'Launch-Pad'
  Im Launch-Pad-Fenster werden vier Menüs angeboten:

Ikon-Menü	Dieses bietet neben den in Pkt. 1. beschriebenen SNiFF-Tools Funktionen an, von denen die Mehrzahl selbsterklärend ist. An dieser Stelle Bemerkungen zu zwei Funktionen: Log ... Anzeige der SNiFF-Logdatei ~/sniff.log Preferences Eröffnung der Präferenz-Dialogboxen (siehe c))
Projekt-Menü	siehe 3.3
Windows-Menü	Anzeige aller geöffneten Fenster
Help-Menü	Online-Hilfe

c) SNiFF-Präferenzen setzen

Mittels Präferenzen lassen sich bestimmte Aspekte und Verhaltensweisen von SNiFF und seiner Tools einstellen. Sie werden an drei Orten mit steigender Priorität abgelegt:

Level	Ort	erstellt/modifiziert	Allgemeinheitsgrad/Priorität (*)
1.	$SNIFF_DIR/SitePrefs.sniff	Erstellung/Modifikation: durch SNiFF - Customizer Lesen: Launch-Pad-Fenster: Ikon-Menü - Preference - Level: Site	Höchster Allgemeinheitsgrad; Zentral für alle SNiFF-Nutzer eingestellt Priorität: niedrigste
2.	~/.UserPrefs.sniff	durch SNiFF-Nutzer: Erstellung: 1. SNiFF-Start Modifikat.: Launch-Pad-Fenster: Ikon-Menü - Prefer. - Level: User	Abweichend von obigen zentralen Einstellungen kann der Nutzer seine allgemeinen SNiFF-Präferenzen Priorität: mittlere
3.	SNiFF-Wurzel/<proj.-dir.>/<proj.>.proj | <proj.>shared (Projektbeschreibungsfile)	SNiFF-Nutzer: Erstellung: Projekt einrichten (siehe 3.3.) Modifikat.: Proj.-Editor-Fenster: Project-Attributes of Checkmarked Projects (siehe 3.3.)	Projektspezifische Präferenzen. Priorität: höchste

(*) Bei Angaben zum gleichen Sachverhalt überschreiben solche mit höherer Priorität die mit niedrige-

rer (keine Angabe: Undefined).

Präferenzen sichten/ändern (Level 1,2):

Die Präferenz-Dialogbox (Launch-Pad-Fenster: Ikon-Menü - Prefer.) verfügt zu oberst über zwei Pop-up-Menüs:

• View
  - General
  - Global GUI ...
  - Debugger
    . . .
• Level
  - Site (Level 1) Quelle/Ziel: SitePrefs.sniff
  - User (Level 2) Quelle/Ziel: .UserPrefs.sniff

Normalerweise sind durch den System-Costumer auf dem Site-Level alle Angaben für Sie akzeptabel voreingestellt, so daß Sie auf dem darunter liegenden User-Level keine Veränderungen mehr vorzunehmen brauchen. (Schauen Sie sich auf dem Site-Level einmal die Einstellungen an, z.B. hinsichtlich des voreingestellten Debuggers.)

d) Einrichten einer Arbeitsumgebung

Benutzen Sie SNiff nur zur Analyse von Sourcen, so benötigen Sie keine Arbeitsumgebung (und Sie können diesen Punkt überspringen). Eine Arbeitsumgebung ist erforderlich, falls Sie die Sourcen auch modifizieren/ausführen wollen (Edit - Compile - Link - Debug - Zyklus). **Dies ist für die Aufgaben im Rahmen der PI3 erforderlich!**

In einer Arbeitsumgebung werden 1 - n Projekte verwaltet. (Prinzipiell sind mehrere Arbeitsumgebungen möglich, wir beschränken uns der Einfachheit halber auf eine.)

Einrichten einer Arbeitsumgebung:

Launch-Pad-Fenster: Ikon-Menü - Preferences - View: General - Level: User:
  Feld: **W**orking **E**nvironment **S**tate **D**irectory: ...
  Button: Directory ...
  Directory Dialog-Box: unter der SNiFF-Wurzel Eingabe des Directory-Namens:
  “SNiFF_WESD” #empfohlener Name
  Create Directory; Select
  Preference-Dialog-Box: Save; Close

Damit wird unter Ihrem sniff-Knoten das Verzeichnis “SNiFF_WESD” mit drei Dateien angelegt, in denen SNiFF Informationen verwaltet (z.B. über alle eingerichteten Projekte (Pkt. 3.3)).

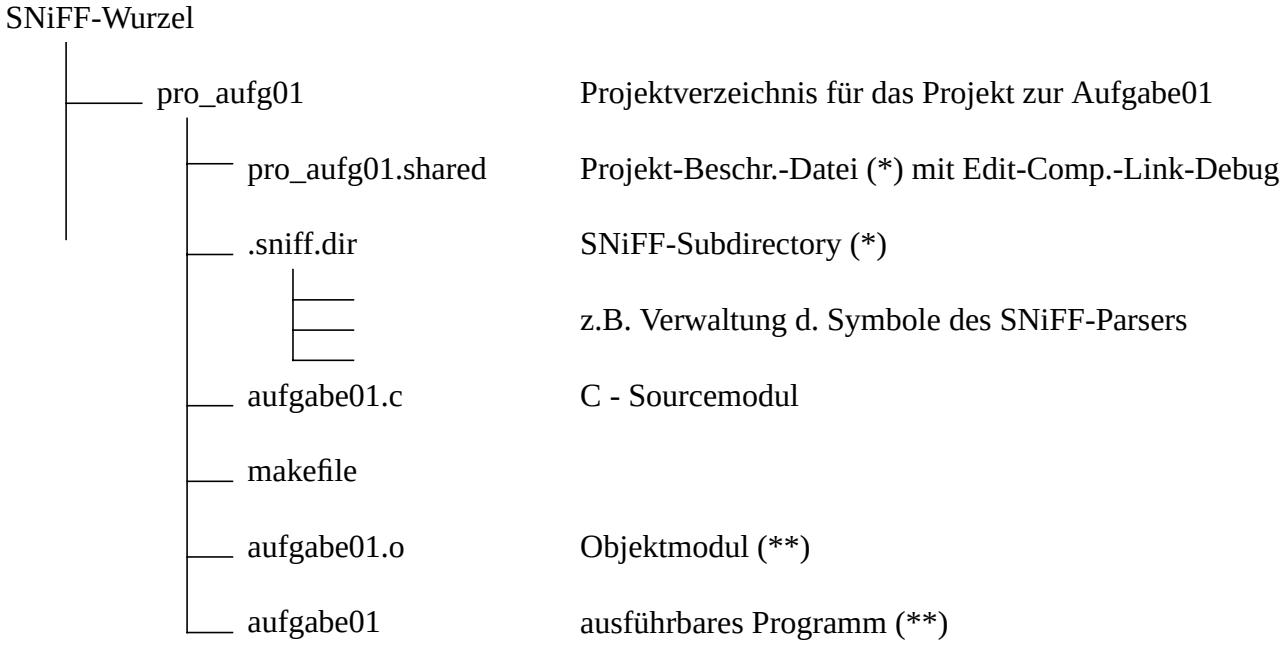
e) SNiFF beenden

Ikon-Menü: Quitt SNiFF+

3.3 Projekte

Für jede Entwicklungsaufgabe müssen Sie ein Projekt unter einem speziellen Verzeichnisknoten einrichten (siehe 3.1.).

Zu einem Projekt gehören im allgemeinen folgende Files bzw. Subdirectories:
Das Beispiel bezieht sich auf eine fiktive Übungsaufgabe 'Aufgabe01'. Diese wird in diesem und in den Folgepunkten weiterverfolgt. Sie beinhaltet einen Edit-Compile-Link-Debug - Zyklus und erfordert somit auch eine Arbeitsumgebung. Die Extension der Projekt-Beschreibungsdatei ist *.shared.



(*) Werden beim Anlegen des Projektes von SNiFF erzeugt

(**) Erzeugt durch das makefile

Ähnlich ist die Struktur bei den Aufgaben zum Pas0-Compiler. So könnten Sie das Projektverzeichnis für die Scanneraufgaben "pro_scan" nennen und hierunter die Projekt-Beschreibungsdatei "pro_scan.shared" und sämtliche Scanner *.c- und *.h-Files führen.

Für den Fall, daß die Sourcen nur analysiert werden (nur Browsen), heißt die Extension der Projekt-Beschreibungsdatei *.proj. Diese Variante wird nachfolgend unter "pro_aufg00 erläutert (Pkt. 3.3.2).

3.3.1 Projekt anlegen/öffnen/schließen/löschen
- mit Edit - Compile - Link - Debug - Zyklus -

Ziel ist es hier, ein Projekt mit einfachen Eigenschaften anzulegen.

a) Projektverzeichnis (mit Sourcen und Makefile) anlegen

Obigem Beispiel folgend:

```
cd ~/sniff
mkdir pro_aufg01
```

Verfügen Sie bereits über Sourcen (*.c, *.h) und/oder das Makefile, so kopieren Sie diese in das Verzeichnis. Ansonsten siehe 3.3.1-d und 3.4.

cp aller Sourcen (*.c, *.h) und des Makefiles in das Verzeichnis

b) Anlegen Projekt-Beschreibungsdatei und SNiFF-Subdirectory

Launch-Pad-Fenster: New Project ...
Eröffnet wird ein Project-Wizard:

- How do you intend to use the new SNiFF+ Project?
  - Standard Setup
  - Next>
- Standard Setup
  Create a new SNiFF+ Project from scratch
  - Next>
- Your development organization
  - No, No, None
    Next>
- Select file types
  - C/C++
  - Next>
- Specify Private Working Environment
  PWE root directory: Browse: bis “pro_aufg01”, Select
  PWE name: pro_aufg01
  Damit ist der Projektname **auf pro_aufg01.shared** festgelegt
  Next>
- Create the new SNiFF+ Project
  Create Subprojects: off
  Create SNiFF+’s Makefiles: off #da eigenes Makefile
  Next>

Project-Summary wird angezeigt
Button: Finish

Nach erfolgreichem Anlegen des Projektes wird das ‘Project-Editorfenster’ für ‘pro_aufg01.shared’ geöffnet. Alle im Verzeichnis bereits enthaltenen Sourcen wurden vom SNiFF-Parser geparst, womit Symbole in die SNiFF-Symboltabelle übernommen wurden und der Sourcecode strukturiert dargestellt wird.

Dieses Parsen geschieht auch dann, falls Sie später weitere Sourcefiles mit dem SNiFF-Editor erfassen (automatisches Parsen bei jedem Save). Ausnahme: Sie kopieren nachträglich auf UNIX-Ebene Sourcen in Ihr Projektverzeichnis. In diesem Fall erfolgt ein nachträgliches parsen durch:
Project-Editor - Fenster: Project - Add/Remove Files to/from ...

c) Projekt öffnen

- Projekt erstmals geöffnet (automatisch nach b)):

  Dem Projekt muß noch das Makefile bekannt gemacht werden:
  Project Editor - Fenster: pro_aufg01.shared
  # make-Angaben
  Project - Attributes of Checkmarked Projects ...

View: Make (Pop-up-Schalter)
Make: Make Command: make -f makefile
Generate Make Support Files: Button: off
Generated Make Macros: leer
Targets: Executable: aufgabe01 #Ziel aus dem Makefile
OK
Damit werden diese Angaben zu projektspezifischen Präferenzen (Level 3).

- Projekt ein weiteres Mal öffnen:

  Falls Sie zwischenzeitlich ein anderes Projekt geöffnet hatten. Entsprechend müssen Sie auch verfahren, fall Sie nach dem Schließen eines Projektes ein anderes eröffnen wollen.

  Setzen der Workspace-Variablen:
  Launch-Pad - Fenster: Ikon-Menü - Preferences ...
  View: General; Level: User
  Default Working Envionment: PWE: pro_aufg01
  Working Environment State Directory: ~/sniff/SNiFF_WESD
  Save; Close

  Open Projekt: pro_aufg01.shared; Open

d) Sourcefiles und Makefile eingeben

Sie könnten nun, falls noch nicht geschehen, Sourcefiles und/oder das Makefile eingeben (Ablauf: siehe 3.4). Wie bereits oben erwähnt, wird dann bei jedem zwischenzeitlichen ‘Save’ der SNiFF-Parser aufgerufen.

e) Projekt schließen/löschen

Projekt im Launch-Pad-Fenster auswählen;
Project - Close/Delete

3.3.2 Projekt anlegen/öffnen/schließen/löschen
- nur Browsen -

Ziel ist es hier, ein Projekt mit einfachen Eigenschaften anzulegen.

a) Projektverzeichnis mit Sourcen und makefile anlegen

Beispiel für eine Aufgabe00:
```
cd ~/sniff
mkdir pro_aufg00
cp aller Sourcen (*.c, *.h) und des Makefiles in das Verzeichnis
```

b) Anlegen Projekt-Beschreibungsdatei und SNiFF-Subdirectory

Launch-Pad-Fenster: New Project ...
Eröffnet wird ein Project-Wizard:

- Select development task
  - Browsing - only Setup
  - Next>
- Select file types
  - C/C++
  - Next>
- Specify project location and name
  - Source code root directory - Browse
    Directory Dialog-Fenster:
    Unter Directory ‘sniff’ den Directory-Namen ‘pro_aufg00’ eingeben und
    Button: Create Directory: ‘pro_aufg00’ wird angelegt
    Unter Directory ‘sniff’ erneut den Directory-Namen ‘pro_aufg00’ eingeben und
    Button: Select: damit ist der Projektname auf ‘**pro_aufg00.proj**’ festgelegt
  - Next>
    Project-Summary wird angezeigt
  - Button: Finish

Nach erfolgreichem Anlegen des Projektes wird das ‘Project-Editorfenster’ für ‘pro_aufg00.proj’ geöffnet. Alle im Verzeichnis bereits enthaltenen Sourcen wurden vom SNiFF-Parser geparst, womit Symbole in die SNiFF-Symboltabelle übernommen wurden und der Sourcecode strukturiert dargestellt wird.

Dieses Parsen geschieht auch dann, falls Sie später weitere Sourcefiles mit dem SNiFF-Editor erfassen (automatisches Parsen bei jedem Save). Ausnahme: Sie kopieren nachträglich auf UNIX-Ebene Sourcen in Ihr Projektverzeichnis. In diesem Fall erfolgt ein nachträgliches parsen durch:
Project-Editor - Fenster: Project - Add/Remove Files to/from ...

c) Projekt öffnen

Falls Sie zwischenzeitlich ein Projekt oder SNiFF schließen, müssen Sie das Projekt öffnen:
Launch-Pad-Fenster: Project - Open;
gewünschtes Projekt-Directory zum Aktuellen machen;
Kursor auf Project-Beschreibungsdatei und Open (bzw. Doppelklick).

d) Projekt schließen/löschen

Projekt im Launch-Pad-Fenster auswählen;
Project - Close/Delete

3.4. Beispiel: Lösung einer Übungsaufgabe

Erläutert wird ein einfacher Gesamtablauf mit Edit - Compile - Link - Debug - Zyklus.

Gegeben: Übungsaufgabe Nr. 01, in der Sie zur Lösung ein C-Sourcecode-File ‘aufgabe01.c‘ eingeben und austesten müssen. Ergebnis soll das ausführbare Programm ‘aufgabe01‘ sein.

Zwei Teilschritte sind erforderlich:

a) Projekt anlegen / öffnen

Der Einfachheit halber nennen Sie auch das Projekt ‘pro_aufg01.shared‘. Das Anlegen dieses Projektes ist in 3.3.1 beschrieben. Das ‘makefile‘ könnte so aussehen:

```
aufgabe01:   aufgabe01.o
   gcc  -o  aufgabe01   aufgabe01.o
aufgabe01.o:   aufgabe01.c
   gcc  -g  -c  aufgabe01.c
```

(Denken Sie daran, daß Kommandozeilen im Anschluß an make-Regeln mit einem <tab>-Zeichen beginnen müssen.)

Abschließendes Ergebnis ist das geöffnete Projekt-Directory-Fenster.

Ist das **Projekt bereits angelegt**, so ist es nur zu öffnen (siehe 3.3.1).

b) Eingabe / Modifikation Sourcefile (Edit), Compile, Link, Debug

• Eingabe Source-File

- Aufruf des Editor-Tools
- Eingabe des Source-Codes
- Zwischenzeitliches Sichern:
  erstmaliges Sichern: File - Save as ...; aufgabe01.c; Save
  weiteres Sichern: File - Save
  Mit dem Sichern wird auch der SNiFF-Parser aufgerufen

• Compile, Link

- Compile / Link starten: Source-Editor-Fenster:
  Target - Make ‘aufgabe01’
- Eröffnung des Shellfensters und Anzeige des Compile-, Link-Protokolls
- Compile-, Link-Zyklus:
  -- Fehler werden mit Zeilennummer angezeigt
  -- Fehlerkorrektur: Source-Editor-Fenster: Positioning; GoTo Line ...
  -- abschließend erneuter Compile-, Link-Zyklus bis der Sourcecode letztendlich fehlerfrei ist

• Debug

- Debugger starten: Source-Editor-Fenster: Debug aufgabe01
- anschließend erfolgt der Aufruf des zu testenden Programmes ‘aufgabe01‘ (über ‘Run‘) und die Testarbeit mit dem Debugger (siehe dazu 2.2.). (Sollte Argumentübergabe an das Programm erforderlich sein, so können Sie diesw in dem “Arguments-Fenster” tun. Z.B. beim Scanner mit “-T <bsp.pas”