Forschungszentrum Jülich

Institut für Chemie und Dynamik der Geosphäre 1:
Stratosphärische Chemie

Entwicklung von Softwarewerkzeugen zur Messdatenorganisation und Datenvisualisierung für die interaktive Programmierumgebung IDL im Institut für Chemie und Dynamik der Geosphäre des Forschungszentrums Jülich

Reimar Bauer

Jül-3786

Entwicklung von Softwarewerkzeugen zur Messdatenorganisation und Datenvisualisierung für die interaktive Programmierumgebung IDL im Institut für Chemie und Dynamik der Geosphäre des Forschungszentrums Jülich

Version 1.0

Reimar Bauer

Berichte des Forschungszentrums Jülich ; 3786
ISSN 0944-2952
Institut für Chemie und Dynamik der Geosphäre 1:
Stratosphärische Chemie Jül-3786

Zu beziehen durch: Forschungszentrum Jülich GmbH · Zentralbibliothek
D-52425 Jülich · Bundesrepublik Deutschland
02461/61-6102 · Telefax: 02461/61-6103 · e-mail: zb-publikation@fz-juelich.de

Abstract

Software tools to organize and visualize scientific data sets using the Interactive Datavisualisation Language (IDL), implemented by the *Institute for Chemistry and Dynamic of the Geosphere* (ICG), *Research Center Jülich*

This booklet describes the principle properties and possibilities of the ICG IDL-library. The base for data exchange between modules and files is the ICG-DATA-STRUCTURE. The setup and definition of this data structure is explained in detail. The data structure includes definitions which are useful for a relational database of scientific data. For visualization purposes of the data a HTML and a PLOT environment is described. The PLOT environment standardizes high quality graphics of scientific data. Possible output devices for the plots are screen, postscript and GIF-Animation. The HTML environment is used as an easy-to-use tool to publish scientific data and graphics in the world wide web.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Beispiele

Kapitel 1

Einleitung

Mit der Einführung des Softwareproduktes IDL der Fa. RSI INC. 1994 zunächst in den Instituten ICG-3 und ICG-1 zur Auswertung und Interpretation von Daten wurde gleichzeitig die gemeinsame Nutzung der entwickelten Software-Werkzeuge geplant. IDL, siehe [*IDL Reference Guide*(1999)] ist eine plattformunabhängige interaktive Datenvisualisierungs- und Programmier-Umgebung. Diese Sprache ist schnell erlernbar und bietet durch die Selbsterweiterung des Sprachumfangs die Möglichkeit ein Produkt zu schaffen, das für individuelle Aufgabenstellungen optimal beschaffen ist.

Bei beiden Instituten werden ähnliche Messmethoden und Analysemethoden sowohl im Experiment- als auch im Modell-Bereich verwendet. Konzeptionell gut erarbeitete Datenverarbeitungs- und Darstellungsprogramme sind in allen Teilbereichen nutzbar.

Die Bibliothek ermöglicht es den Wissenschaftlern während einer Messkampagne mit einem minimalen Zeitaufwand Daten auszuwerten und darzustellen und danach diese zu veröffentlichen. Mit Hilfe der Bibliothek sind schon mehr als 100 Vorträge, Veröffentlichungen und Poster erstellt worden.

Der erreichte Umfang der Bibliothek, ist der Anlass für diese Beschreibung, wobei weitere Ergänzungen oder Verbesserungen nicht ausgeschlossen werden.

1.1 Konzepte und Autoren

Die Bibliothek wurde von Wissenschaftlern und technischen Angestellten erarbeitet, derzeit mitwirkende Autoren an der ICG-Bibliothek sind:

von links nach rechts: Frank Holland, Reimar Bauer, Nicole Thomas, Marlene Busch, Franz Rohrer, Theo Brauers und Jürgen Ankenbrand.

Abbildung 1.1: Autoren der Bibliothek

Durch die IDL_SOURCE-Arbeitsgruppe, bestehend aus Reimar Bauer (RB), Theo Brauers (TB), Frank Holland (FH) und Franz Rohrer (FR), sind folgende Konzepte definiert und programmiert worden.

- Datenausgabe in HTML, Tabellarische Übersichten von Strukturen und Tabellen (TB, FR, RB), siehe Kapitel 3 auf Seite 15.
- Defintion der ICG-Daten-Struktur (RB), siehe Kapitel 4 auf Seite 25. Durch die zunehmende lokale Bearbeitung von Daten ergab sich die Notwendigkeit der Definition einer Datenstruktur für die Verarbeitung von Daten. Diese Datenstruktur vereinfacht den Austausch von Daten über mehrere Programmmodule hinaus, das Lesen und Schreiben verschiedener Datendateiformate, sowie das Darstellen der Daten. Zu diesem Zweck wurde 1998 die ICG-Daten-Struktur in Anlehnung an das netCDF-Datenformat, siehe [*R. Rew et al.*(1992)] definiert. Diese Struktur ermöglicht es, komplexe Datensachverhalte in einem Objekt zusammenzufassen. In den Jahren 1998 bis 2000 sind viele Routinen zum Verarbeiten dieser ICG-Daten-Struktur in der Bibliothek zur Verfügung gestellt worden.
- Anwendungsprogramme für die ICG-Daten-Struktur (RB, FH, TB, FR), siehe Kapitel 4.3 auf Seite 50.

- Katalog der IDL-Bibliotheken (RB), siehe Kapitel 2.3 auf Seite 8. Damit über die verfügbaren Routinen eine Übersicht möglich ist, werden alle Routinen täglich katalogisiert und in Form von HTML-Seiten zur Verfügung gestellt. Da die Bibliothek evtl. geringfügigen Änderungen unterliegt, wird die Arbeitsbibliothek jeden Tag gesichert. Dadurch ist jede Änderung nachvollziehbar.
- Definition der script-orientierten PLOT-Umgebung (FR), siehe Kapitel 5 auf Seite 55.
- Anwendungsprogramme für die script-orientierte PLOT-Umgebung (FR, RB, JA), siehe Kapitel 5.4 auf Seite 81.
- Graphische Benutzerschnittstellen (widgets) für die PLOT-Umgebung (RB), siehe Kapitel 5.3 auf Seite 72.
- Interaktive PLOT-Umgebung (FH, TB)
- Struktur-Operationen auf jeder Struktur-Ebene, Löschen, Umbenennen, Ersetzen, Hinzufügen von Daten oder Strukturen (FH, RB), siehe Kapitel 4.2.1 auf Seite 43.

Derzeit sind die beiden Graphikpakete für die script orientierte Darstellung und die interaktive Darstellung von Daten voneinander getrennt. Mit der Einführung von IDL5.4 werden wir in die Lage versetzt, diese beiden Pakete in einem neuen Grafikpaket zu vereinen.

Nicole Thomas und Jürgen Ankenbrand haben viele der Routinen aus der idl_model Bibliothek geschrieben. Diese Routinen sind für die Visualisierung unserer Modell-Daten bzw. deren Steuerung nötig.

Marlene Busch hält die IDL-Fort-Bildungs-Seminare im Forschungszentrum, unter Berücksichtigung unserer Routinen. In verschiedenen Projekten wie z.B. die Einführung von ION (IDL On the Net) ist sie für uns eine wertvolle Unterstützung.

Kapitel 2

Konfiguration der ICG-IDL Bibliothek

2.1 Die ICG-IDL Bibliothek

Seit 1994 ist eine Vielzahl von Modulen erstellt worden. Diese Module ermöglichen es einem Benutzer, durch Kombination der Module ein neues Modul oder ein grösseres Programm zu erstellen. Die Routinen sind in der IDL Bibliothek des ICGs (siehe [*R. Bauer et al.* (2000)]) organisiert.

Das nachfolgende Diagramm gibt die Organisation der Bibliothek und den bei allen Nutzern eingestellten Suchpfad von IDL wieder.

`+../rsi/idl/lib`	mit allen Unterverzeichnissen
`+../idl_source/idl_work/aaa`	
`+../idl_source/idl_lib/aaa`	
`+../idl_source/idl_links`	
`+../idl_source/idl_work`	mit allen Unterverzeichnissen
`+../idl_source/idl_lib`	mit allen Unterverzeichnissen
`+../idl_source/idl_experiment`	mit allen Unterverzeichnissen
`+../idl_source/idl_model`	mit allen Unterverzeichnissen

Tabelle 2.1: Konfiguration: Pfad Definition

Routinen, die im ICG entwickelt wurden, befinden sich in den Verzeichnissen `idl_work`, `idl_experiment` und `idl_model`.

Das Verzeichnis `../idl_work/aaa` ist dem ganzen Suchpfad der Bibliotek übergeordnet. In diesem Verzeichnis sind alle Entwickler schreibberechtigt. Wird eine Routine aus der Bibliothek eines gerade nicht greifbaren Entwicklers abgeändert, kann sie von einem anderen dort abgelegt und somit ganz vorn im Suchpfad wirksam werden.

Im Verzeichnis `../idl_source/idl_links` sind alle aktuell verwendeten Routinen als `link` auf die entsprechende Routine abgebildet. Dadurch wird der Suchprozess erheblich beschleunigt.

Im Verzeichnis `../idl_source/idl_lib` sind Bibliotheken von externen Autoren organisiert. Dort ist `../idl_lib/aaa` übergeordnet.

Derzeit werden von nachfolgenden Autoren externe Bibliotheken verwendet:

- Ray Sterner, siehe [*R. Sterner et al.*(1992)]
- Craig Markwardt, siehe [*C. Markwardt* (1997)]
- Steve Gaines, siehe [*S. E. Gaines et al.*(1990)]
- Martin Schultz, siehe [*M. Schultz* (2000)]
- Wayne Landsman, siehe [*W. Landsman* (1997)]
- Marc Davis, siehe [*M. Davis et al.* (1998)]

In `idl_work` sollen ausschliesslich allgemeingültige Routinen gespeichert werden, in `idl_experiment` bzw. `idl_model` spezifische Experimente-Routinen bzw. Modell-Routinen.

Die Zugriffsrechte auf Dateien unterhalb des Verzeichnisses `../idl_source` werden zu jeder vollen 5. Minute für alle auf Lesen gesetzt.

Die ICG-Bibliothek unterliegt der GNU-Public-Library Lizenz. Die Lizenz- und Urheberrechte der externen Autoren sind auf deren WEB Seiten definiert. Die Routinen der Bibliothek werden täglich durch einen HTML-Katalog zusammengefasst und dargestellt.

2.2 Initialisierung

Damit auf die Routinen von verschiedenen Plattformen in einer definierten Weise zugegriffen wird, wurde eine Initialisierungs-Datei und eine Umgebungsvariable definiert. Die Umgebungsvariable `IDL_SOURCE` muss auf den Pfad von `idl_source` zeigen. Durch die Initialisierungs-Datei `idl.all` in `idl_source` wird der Suchpfad definiert und einige Voreinstellungen getroffen. Der Suchpfad von IDL setzt sich aus dem lokalen Pfad und dem Bibliothekenpfad zusammen. Eine Routine wird von IDL zuerst in dem lokalen Pfad (`CD, current=curr & PRINT, curr`), dann in `!PATH` gesucht.
Die Definition mit einer idl_startup Datei sieht z.B. wie folgt aus.

```
@/usr/local/icg/icg/idl_source/idl.all

window,0,xs=100,ys=100,color=256
wdelete,0
```

Beispiel 2.1: Konfiguration: idl_startup.pro

```
DEFSYSV, '!icg', $
   CREATE_STRUCT(NAME='!ICG', 'USER', '', 'HOST', '', 'IDL_SOURCE', '', 'IDLDE', '', 'SEP', '' $
      , 'A', ['/idl_work/aaa', '/idl_lib/aaa','/idl_links','/idl_work', '/idl_lib', $
      '/idl_experiment', '/idl_model','/easy/program/idl', '/idl_obsolete'] )

; seperator definition
IF !version.os_family EQ 'unix' THEN !icg.sep =':'
IF !version.os_family EQ 'Windows' THEN !icg.sep =';'

; host and user
IF !version.os_family EQ 'unix' THEN BEGIN & SPAWN,'whoami', b & !icg.user = b & ENDIF
IF !version.os_family EQ 'unix' THEN BEGIN & SPAWN,'hostname', b & !icg.host = b & ENDIF
IF !version.os_family EQ 'Windows' THEN !icg.user = GETENV('username')
IF !version.os_family EQ 'Windows' THEN !icg.host = GETENV('computername')
!icg.user=STRLOWCASE(!icg.user)
!icg.host=STRLOWCASE(!icg.host)

; idl source
!icg.idl_source=GETENV('IDL_SOURCE')
IF !icg.idl_source EQ '' THEN PRINT,'DEFINE IDL_SOURCE ENVIRONMENT VARIABLE, STANDARD PARAMETERS WILL BE USED'

IF !version.os_family EQ 'unix' AND  !icg.idl_source EQ '' THEN !icg.idl_source ='/usr/local/icg/icg/idl_source'
IF !version.os_family EQ 'Windows' AND  !icg.idl_source EQ '' THEN !icg.idl_source='S:\links\idl_source'

; idlde
IF !version.os_family EQ 'Windows' THEN !icg.idlde=GETENV('IDLDE')
IF !icg.idlde EQ '' THEN !icg.idlde=!dir

; grafics
IF !version.os_family EQ 'Windows' then DEVICE, DECOMPOSED=0
if strlowcase(!version.os) eq 'linux' then device,decomposed=0
if strlowcase(!version.os) eq 'linux' then window,0,xsize=20,ysize=20,color=256
if strlowcase(!version.os) eq 'linux' then wdelete,0

!p.background=255
!p.color=0

IF !version.os_family EQ 'unix' then device, pseudo_color = 8
IF !version.os_family EQ 'unix' THEN device, retain = 2

; expand path
!icg.a = !icg.idl_source + !icg.a
IF !version.os_family EQ 'unix' THEN  b='+/usr/local/idl/idl/lib'
IF !version.os_family EQ 'Windows' THEN  b=!icg.idlde+'/lib'

; add the icg1 lib for all ICG-1 users
IF !version.os_family EQ 'unix' AND strpos(!icg.user,'icg1') gt -1 then $
   b = b + !icg.sep+ '+/usr/local/icg/icg1/idl'


IF !version.os_family EQ 'Windows' AND strpos(!icg.user,'icg1') gt -1 then $
   b = b + !icg.sep+ '+S:/links/icg1/idl'

; loop thru all path items
FOR i=0, N_ELEMENTS(!icg.a)-1 DO b=b+!icg.sep+'+'+!icg.a[i]
; expand the path
b = EXPAND_PATH(b)

; Add to existing path
!path = !path+!icg.sep+b

; get rid of used variables
DELVAR, b, i

;default widget font
;IF !VERSION.OS_FAMILY eq 'unix' then widget_control,default_font='-adobe-helvetica-medium-r-normal--12-*-*-*-*-60-*-*'
if strlowcase(!version.os) eq 'linux' then widget_control,default_font='-adobe-helvetica-medium-r-normal--12-*-*-*-*-60-*-*'
IF !VERSION.OS_FAMILY eq 'Windows' then widget_control,default_font='HELVETICA*VARIABLE*14'

; print an informational message
PRINT, 'Your !PATH settings have be modified by idl.all (Wed May 10 10:49:00 2000)'
PRINT, '-----'
PRINT, 'Welcome '+!icg.user+'@'+!icg.host+' to IDL Version: ' + !version.release+' !'
PRINT, 'Current time: '+SYSTIME()
```

Abbildung 2.1: Konfiguration: idl.all (Initialisierungs Datei)

2.3 Katalog

Auf Grund der vielen Routinen, die im Laufe mehrerer Jahre durch verschiedene Programmierer entstehen, soll ein Katalogisierungswerkzeug einen Überblick über vorhandene Projekte und Routinen verschaffen. Dieses Werkzeug stellt verschiedene Sichten der aktuellen Bibliothek in HTML zur Verfügung.

2.3.1 Voraussetzungen:

Um die Katalogisierung möglichst effizient zu gestalten, müssen alle Routinen einen standardisierten Header enthalten. Dieser Header ermöglicht auch die Zuordnung der jeweiligen Routine zu übergeordneten Kategorien.

Definition des Headers

```
; Copyright (c) YYYY, Forschungszentrum Juelich GmbH ICG
; All rights reserved.
; Unauthorized reproduction prohibited.
; This software may be used, copied, or redistributed as long as it is
; not sold and this copyright notice is reproduced on each copy made.
; This routine is provided as is without any express or implied
; warranties whatsoever.
;
;+
; NAME:
;   ROUTINE_NAME
;
; PURPOSE:
;   Tell what your routine does here.  I like to start with the words:
;   "This function (or procedure) ..."
;   Try to use the active, present tense.
;
; CATEGORY:
;   Put a category here.  For example:
;   Widgets.
;
; CALLING SEQUENCE:
;   Write the calling sequence here. Include only positional parameters
;   (i.e., NO KEYWORDS). For procedures, use the form:
;
;   ROUTINE_NAME, Parameter1, Parameter2, Foobar
;
;   Note that the routine name is ALL CAPS and arguments have Initial
;   Caps.  For functions, use the form:
;
;   Result = FUNCTION_NAME(Parameter1, Parameter2, Foobar)
;
;   Always use the "Result = " part to begin. This makes it super-
;   obvious to the user that this routine is a function!
;
; INPUTS:
;   Parm1:  Describe the positional input parameters here. Note again
;       that positional parameters are shown with Initial Caps.
;
; OPTIONAL INPUTS:
;   Parm2:  Describe optional inputs here. If you don't have any, just
;       delete this section.
;
; KEYWORD PARAMETERS:
;   KEY1:   Document keyword parameters like this. Note that the
;        keyword is shown in ALL CAPS!
;
```

```
;   KEY2:   Yet another keyword. Try to use the active, present tense
;       when describing your keywords.  For example, if this keyword
;       is just a set or unset flag, say something like:
;       "Set this keyword to use foobar subfloatation. The default
;        is foobar superfloatation."
;
; OUTPUTS:
;   Describe any outputs here.  For example, "This function returns the
;   foobar superflimpt version of the input array."  This is where you
;   should also document the return value for functions.
;
; OPTIONAL OUTPUTS:
;   Describe optional outputs here.  If the routine doesn't have any,
;   just delete this section.
;
; COMMON BLOCKS:
;   BLOCK1: Describe any common blocks here. If there are no COMMON
;       blocks, just delete this entry.
;
; SIDE EFFECTS:
;   Describe "side effects" here.  There aren't any?  Well, just delete
;   this entry.
;
; RESTRICTIONS:
;   Describe any "restrictions" here.  Delete this section if there are
;   no important restrictions.
;
; PROCEDURE:
;   You can describe the foobar superfloatation method being used here.
;   You might not need this section for your routine.
;
; EXAMPLE:
;   Please provide a simple example here. An example from the
;   DIALOG_PICKFILE documentation is shown below. Please try to include
;   examples that do not rely on variables or data files that are not
;   defined in the example code. Your example should execute properly
;   if typed in at the IDL command line with no other preparation.
;
;   Create a DIALOG_PICKFILE widget that lets users select only files
;   with the extensions 'pro' and 'dat'.  Use the 'Select File to Read'
;   title and store the name of the selected file in the
;   variable F.  Enter:
;
;       F = DIALOG_PICKFILE(/READ, FILTER = ['pro', 'dat'])
;
; MODIFICATION HISTORY:
;   Written by:     Your name here, Date.
;   July, 1994      Any additional mods get described here.
;           Remember to change the stuff above if you add a new keyword
;            or something!
;-
```

Beispiel 2.2: Konfiguration: Beispiel Header

Definitionen der Kategorien für IDL WORK

Damit die Routinen leichter zu ordnen und so schneller zu finden sind, werden sie in Kategorien unterteilt. Es darf nur genau eine Kategorie für jede Routine verwendet werden.

CASE_TOOLS	Software-Entwicklungswerkzeuge (Wartungshilfen)
DATAFILES/FILE	Verschiedene I/O-Routinen
DATAFILES/FILTER	Konvertierungsfilter
DATAFILES/MFD	Routinen für das MFD Format
DATAFILES/NASA_AMES	Routinen für die NASA Ames Formate behandeln
DATAFILES/NETCDF	Routinen für das netCDF Format
DATAFILES/ZCHN	Routinen, die das ICG-3 ZCHN/ENZ Daten Format behandeln
ICG_STRUCT	Allgemeine Routinen für die ICG-Daten-Struktur
ICG_STRUCT/DATAFILES/NETCDF	Datenverarbeitungsroutinen für die ICG-Daten-Struktur und netCDF
ICG STRUCT/MATH	Routinen die die ICG-Daten-Struktur verarbeiten. Hier mathematische Routinen
ICG_STRUCT/PROG_TOOLS	Routinen die die ICG-Daten-Struktur verarbeiten. Hier Routinen zur allgemeinen Programmierung
ICG_STRUCT/WIDGETS	widgets für die Verarbeitung der ICG-Daten-Struktur.
MATH	mathematische Routinen
OBSOLETE	überflüssige Routinen, die durch andere ersetzt wurden
PLOT	Verschiedene plot Routinen
PLOT/PLOT2D	2D Darstellungs Routinen (script orientiert)
PLOT/PLOTXY	XY Darstellungs Routinen (script orientiert)
PLOT/MPLOT	plot Routinen (interaktiv)
PROG_TOOLS	Allgemeine Routinen
PROG_TOOLS/STRINGS	String Routinen
PROG_TOOLS/STRUCTURES	Routinen für Strukturen
PROG_TOOLS/WIDGETS	widgets Routinen
TIME	Zeitverarbeitungs Routinen

Tabelle 2.2: Konfiguration: Kategorienbezeichner der Routinen

2.3.2 Katalog der Routinen

Der Katalog ist ein IDL-Programm, das alle IDL-Routinen katalogisiert und dabei auswertet. Die Ergebnisse werden in verschiedenen HTML-Sichten gespeichert. Das Programm wurde 1996 entwickelt und verwendet für die Seitensteuerung Javascript, siehe [*A. Danesh et al.*(1996)]. Über den Katalog erhält der Benutzer Information über jedes Modul:

- Was bewirkt es?
- Welche Übergabeparameter werden benötigt?
- Welche anderen Module benutzt es?

Darüber hinaus erlaubt der Katalog einen schnellen Zugriff auf den Quelltext des Moduls.

Im Nachfolgenden werden einige Ansichten des Kataloges dargestellt.

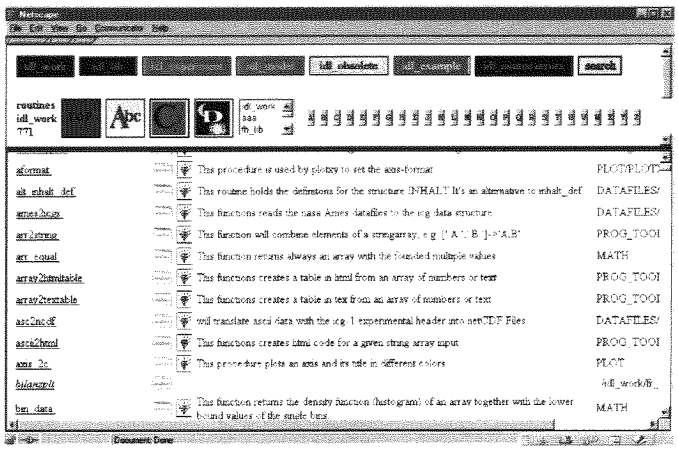


Abbildung 2.2: Katalog: Standardansicht: Blick auf idl_work, alphabetische Anordnung

Abbildung 2.3: Katalog: Aufrufoptionen einer Routine, wird beim Überstreichen des Routinennamens mit dem Mauszeiger durch das *javascript overlib.js* (siehe [*E. Bosrup* (2000)]) sichtbar.

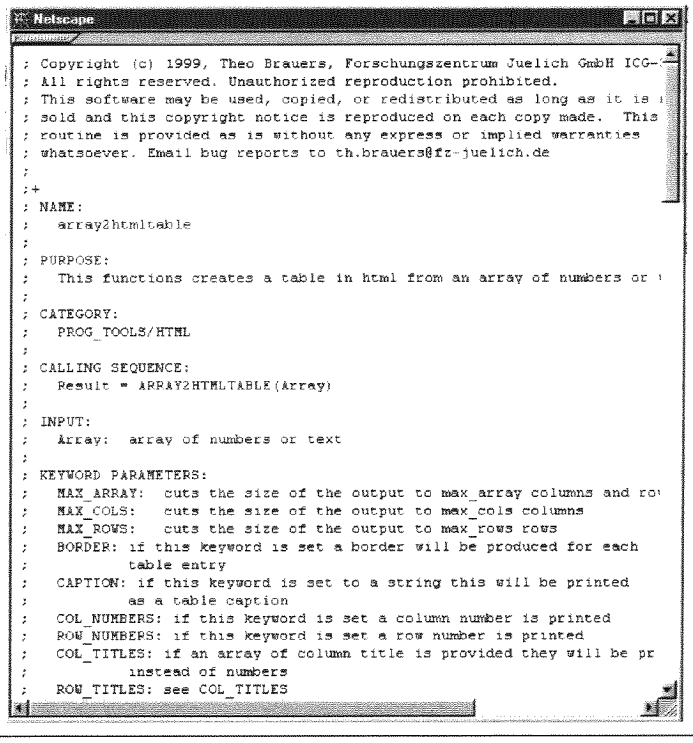


Abbildung 2.4: Katalog: Quelltext einer Routine, wird beim Anwählen des Routinennamens sichtbar

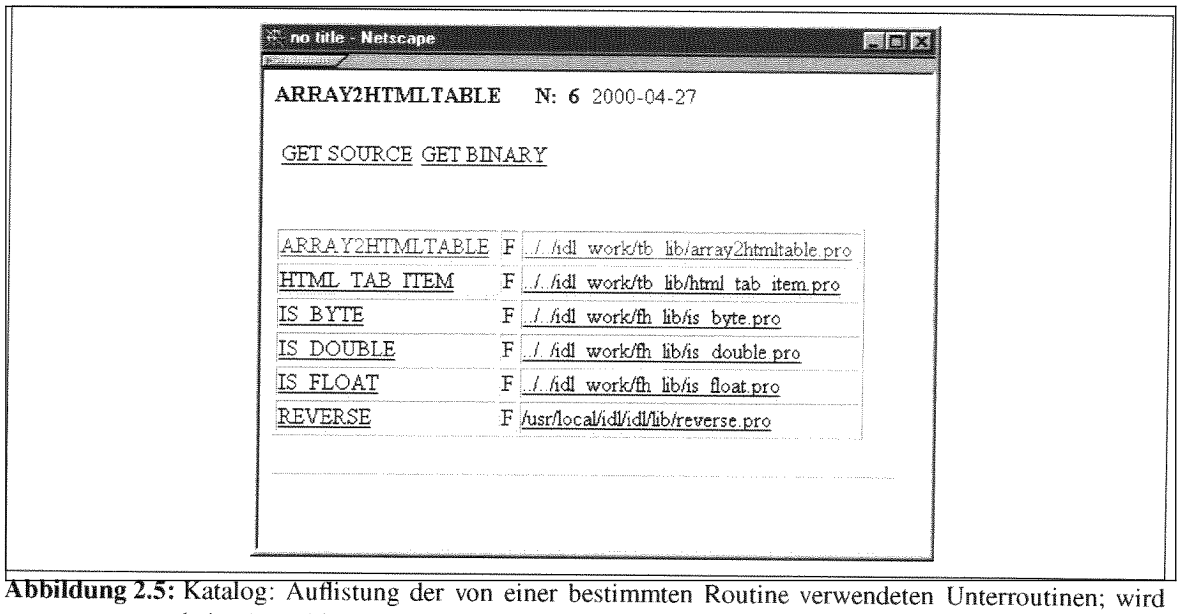


Abbildung 2.5: Katalog: Auflistung der von einer bestimmten Routine verwendeten Unterroutinen; wird beim Anwählen des Routine-Symbols sichtbar

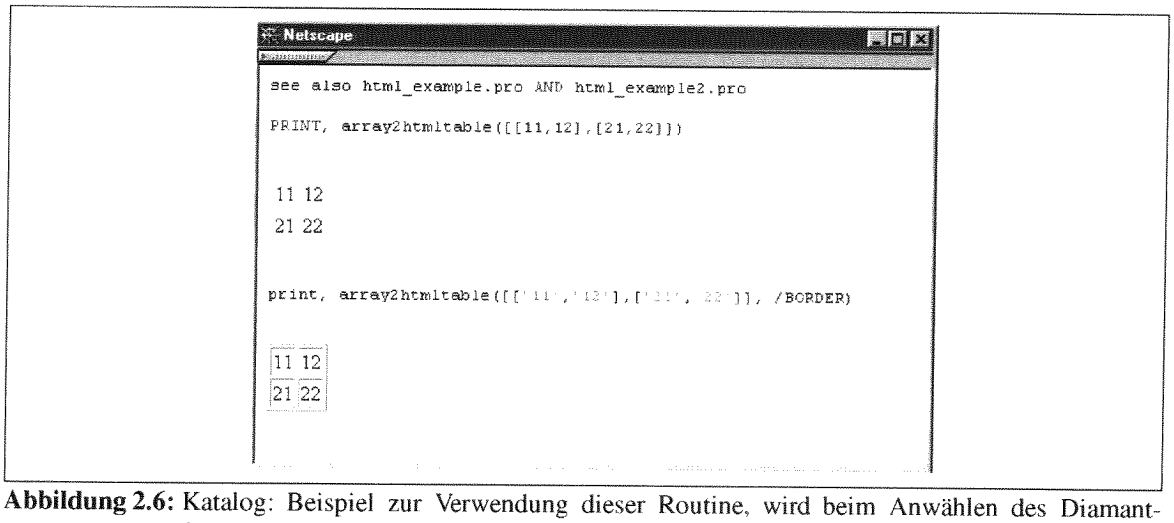


Abbildung 2.6: Katalog: Beispiel zur Verwendung dieser Routine, wird beim Anwählen des Diamant-Symbols sichtbar.

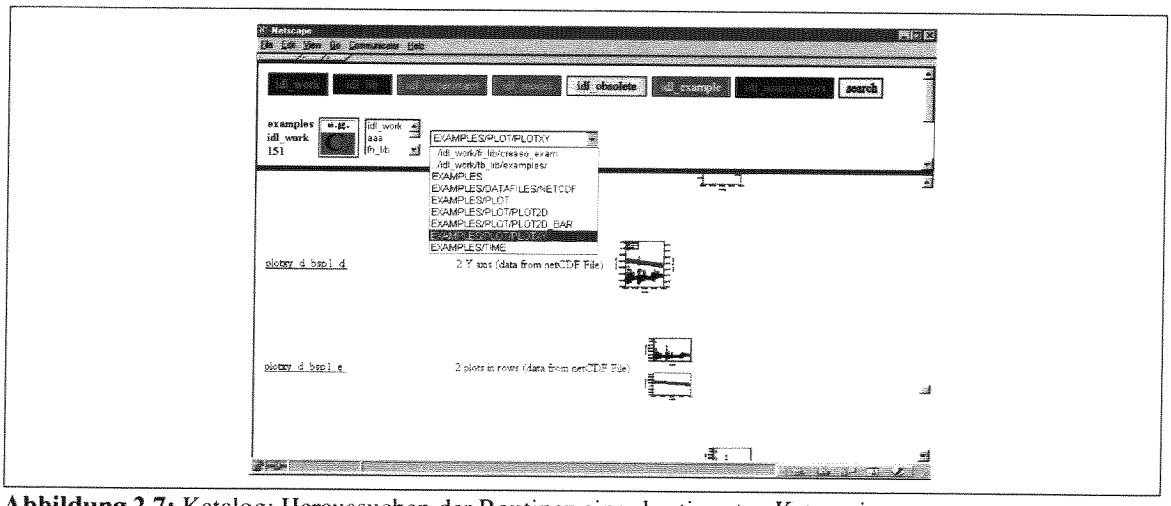


Abbildung 2.7: Katalog: Heraussuchen der Routinen einer bestimmten Kategorie.

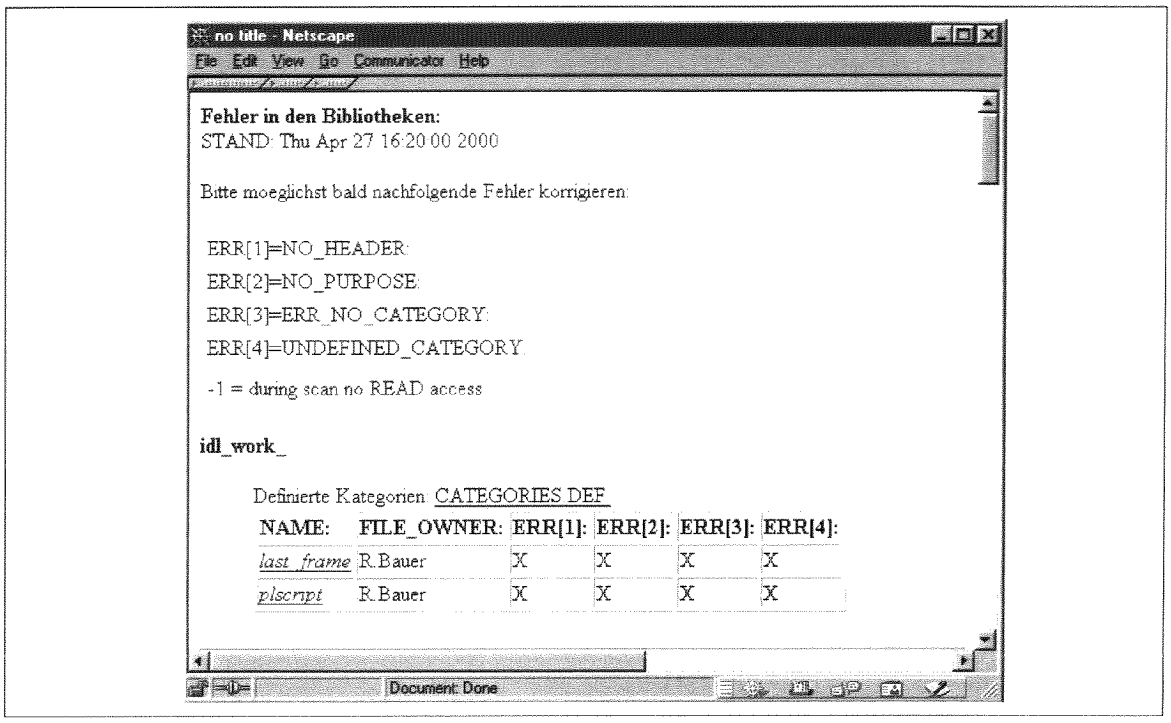


Abbildung 2.8: Katalog: Neben der Katalogansicht erstellt das Programm auch eine Liste der fehlerhaften Routinen, bei denen z.B. Fehler im Header festgestellt wurden

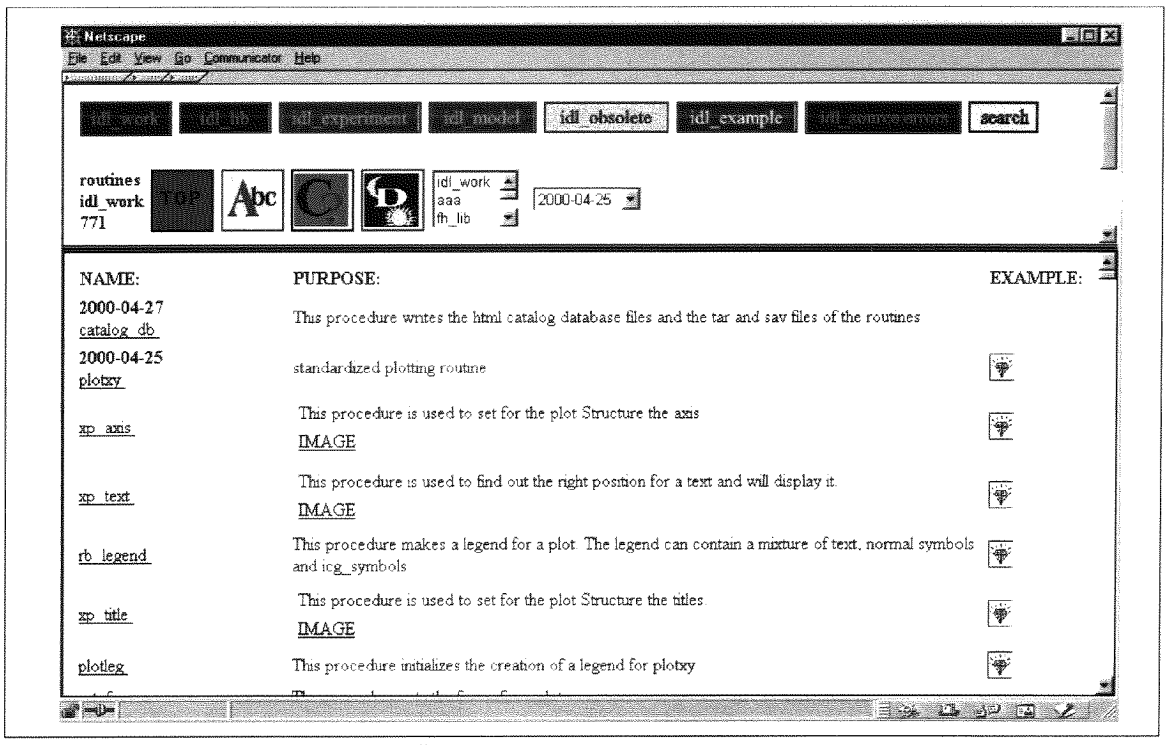


Abbildung 2.9: Katalog: Historie der Änderungen an der Bibliothek

Kapitel 3

Ausgabe von Daten in HTML

Mit dem Katalog, siehe Kapitel 2.3.2 auf Seite 11 wurde schon eine Anwendung von HTML gezeigt. Eine weitere Anwendung liegt in der Visualisierung von Datenarchiven, siehe Kapitel 6 auf Seite 125 bzw. der Visualisierung von Zahlen, die in Datenstrukturen enthalten sind, siehe Kapitel 4.1.6 auf Seite 42.

Im Gegensatz zu herkömmlichen Medien bietet HTML keine Begrenzung in der Seitenbreite oder Seitenlänge. Dadurch ist man in der Lage, auch komplexe Daten darzustellen. Darüberhinaus sind Daten, die in HTML visualisiert werden, in dieser Form plattformunabhängig mit jedem Browser darstellbar.

3.1 Basisbefehle

Eine HTML Datei enthält eine Start- (`html_header`) und eine Ende-Befehlskette (`html_trailer`). Dazwischen befindet sich der eigentliche Text, der dargestellt werden soll.

Diese unterschiedlichen Teile einer HTML-Datei können durch praktische IDL-Routinen modular erzeugt werden. Das Zusammenfügen dieser Module ergibt dann eine lauffähige HTML-Datei. Das nachfolgende Beispiel zeigt die Anwendung der Module.

```
PRO html1
  txt=html_header()
  txt=[txt,array2htmltable(findgen(2,2))]
  txt=[txt,html_trailer()]
  put_file,'test1.html',txt
END
```

Beispiel 3.1: HTML: Anwendung der Module

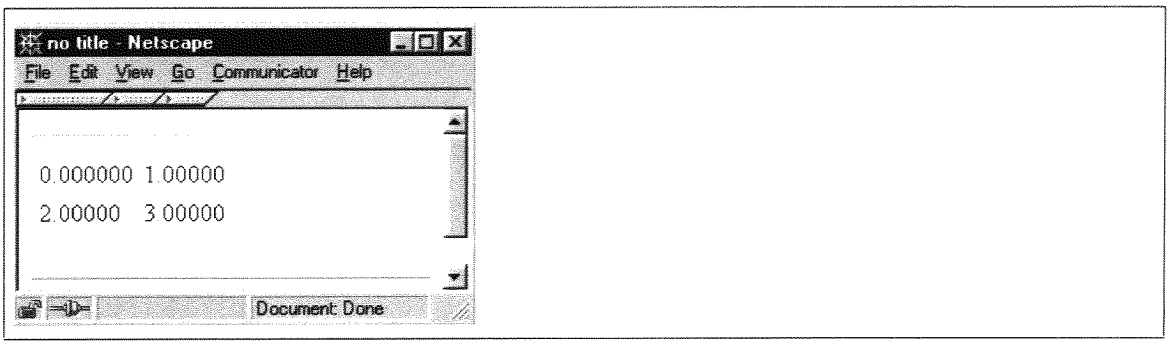


Abbildung 3.1: HTML: Anwendung der Module

3.1.1 html_header()

Diese Funktion erstellt die Start-Befehlskette einer HTML Datei. Diese enthält einen Titel und einige andere Informationen

```
printtab, html_header()

<!-- Thu May 11 08:47:40 2000 / icg@ICG1 -->
<HTML>
<HEAD>
  <TITLE> no title </TITLE>
  <META Name="'generator"' Content="'IDL/html_header.pro"'>
  <META Name="'date"' content="'Thu"' May 11 08:47:40 2000"'>
  <META HTTP-EQUIV="'Content-Type"' Content="'text/html;
   charset=iso-8859-1"'>
</HEAD>

<BODY TEXT="'#000000"' BGCOLOR="'#FFFFFF"' LINK="'#0000FF"'
  VLINK="'#00FF00"' ALINK="'#FF0000"'>
<HR>
```

Beispiel 3.2: HTML: html_header()

3.1.2 html_trailer()

Diese Funktion erstellt die Ende-Befehlskette einer HTML-Datei.

```
printtab, html_trailer()

<HR WIDTH="'100%"'>
<UL>
</UL></BODY></HTML>
```

Beispiel 3.3: HTML: html_trailer()

3.1.3 html_form_alph()

Diese Funktion erstellt den HTML-FORM Code zum Erstellen einer alphabetischen Liste.

```
print, html_form_alph(['ALPHA','BETA','ROMEO'])

FORM>
<TABLE>
<TT><TR>
<TD><TT><INPUT TYPE=button NAME="'ALPHA"' VALUE="'ALPHA"'
  onClick=doALPH(this.value)></TT></TD>
<TD><TT><INPUT TYPE=button NAME="'BETA"' VALUE="'BETA"'
  onClick=doALPH(this.value)></TT></TD>
<TD><TT><INPUT TYPE=button NAME="'ROMEO"' VALUE="'ROMEO"'
  onClick=doALPH(this.value)></TT></TD>
</TR>
</TT></FONT>
</TABLE>
</FORM>
```

Beispiel 3.4: HTML: html_form_alph()

3.1.4 html_form_droplist()

Diese Funktion erstellt den HTML-FORM Code zum Erstellen einer Drop-Liste.

```
print, html_form_droplist(['ALPHA','BETA','ROMEO'])

<FORM NAME="'DROPLIST"'>
<SELECT NAME="'Ziel"' SIZE=1 onChange="'category_calc()"'>
<OPTION>ALPHA</OPTION>
<OPTION>BETA</OPTION>
<OPTION>ROMEO</OPTION>
</SELECT>
</FORM>
```

Beispiel 3.5: HTML: html_form_droplist()

3.1.5 array2htmltable()

Diese Funktion erstellt eine Tabelle in HTML für ein Feld von Zahlen oder Text.

```
printtab, array2htmltable(findgen(10))

<TABLE>
<TR><TD>0.000000</TD></TR>
<TR><TD>1.00000</TD></TR>
<TR><TD>2.00000</TD></TR>
<TR><TD>3.00000</TD></TR>
<TR><TD>4.00000</TD></TR>
<TR><TD>5.00000</TD></TR>
<TR><TD>6.00000</TD></TR>
<TR><TD>7.00000</TD></TR>
<TR><TD>8.00000</TD></TR>
<TR><TD>9.00000</TD></TR>
</TABLE>
<BR>
```

Beispiel 3.6: HTML: array2htmltable()

3.1.6 ascii2html()

Diese Funktion erstellt aus einem ASCII Text einen Text in HTML.

```
printtab,ascii2html('Das ist ein Test')
Das ist ein Test<BR>
```

Beispiel 3.7: HTML: ascii2html()

3.1.7 struct2htmltable()

Diese Funktion erstellt eine Tabelle in HTML von den Tag-Namen einer Struktur.

```
printtab, struct2htmltable({A:10,B:5})

<TABLE>
<TR>
<TD><B>A</B></TD>
<TD><B>B</B></TD>
</TR>
<TR><TD>10</TD><TD>5</TD>
</TABLE>
<BR>
```

Beispiel 3.8: HTML: struct2htmltable()

```
printtab, struct2htmltable({A:10,B:findgen(5),C:{k:findgen(10),$
          1:sindgen(10)}},/border,/subtables,/blockquote)

<TABLE BORDER>
<A NAME="'UDEC809C09580297990000002"'></A>
<TR>
<TD><B>A</B></TD>
<TD><B>B</B></TD>
<TD><B><A HREF="'#TDEC809C09580297990000002"'>
  C</A></B></TD>
</TR>
<TR><TD>10</TD><TD>0.000000</TD><TD><A
 HREF="'#TDEC809C09580297990000002"'>str[1]</A></TD>
<TR><TD></TD><TD>1.00000</TD><TD></TD>
<TR><TD></TD><TD>2.00000</TD><TD></TD>
<TR><TD></TD><TD>3.00000</TD><TD></TD>
<TR><TD></TD><TD>4.00000</TD><TD></TD>
</TABLE>
<BR>

<A NAME="'TDEC809C09580297990000002"'></A><BR>
  <BLOCKQUOTE>
<TABLE BORDER>

<TR><TD COLSPAN="'2"'><FONT SIZE=+2></B><A
 HREF="'#UDEC809C09580297990000002"'>C</A></B>
 </FONT></TD></TR>
<TR>
<TD><B>K</B></TD>
<TD><B>L</B></TD>
</TR>
<TR><TD>0.000000</TD><TD>0</TD>
<TR><TD>1.00000</TD><TD>1</TD>
<TR><TD>2.00000</TD><TD>2</TD>
<TR><TD>3.00000</TD><TD>3</TD>
<TR><TD>4.00000</TD><TD>4</TD>
<TR><TD>5.00000</TD><TD>5</TD>
<TR><TD>6.00000</TD><TD>6</TD>
<TR><TD>7.00000</TD><TD>7</TD>
<TR><TD>8.00000</TD><TD>8</TD>
<TR><TD>9.00000</TD><TD>9</TD>
</TABLE>
<BR>

</BLOCKQUOTE>
```

Beispiel 3.9: HTML: struct2htmltable() mit Optionen

3.1.8 gif2html()

Mit dieser Funktion wird ein Bildschirmausdruck in eine GIF-Datei umgeleitet. Der für die HTML-Darstellung benötigte IMG-TAG wird zurückgegeben.

```
printtab, gif2html('img1')

<TABLE>
<TR><TD><IMG SRC="'file:img1"'></TD></TR>
</TABLE>
<BR>
```

Beispiel 3.10: HTML: gif2html()

Kombination gif2html und array2htmltable

Alle Ausgabe-Module können miteinander kombiniert werden. Das nachfolgende Beispiel beschreibt die Anordnung der benötigten Befehle um eine Tabelle zusammen mit einem Bild auf einer HTML-Seite auszugeben.

```
PRO html1
   file='idl_export.html'
   Z=DIST(20)
   SURFACE,z
   put_file,file,html_header()
   put_file,file,array2htmltable(z,/border,caption='Z='),/append
   put_file,file,gif2html('test.gif',/border,caption='SURFACE,Z'),$
      /append
   put_file,file,html_trailer(),/append
END
```

Beispiel 3.11: HTML: Tabelle und Bild

Z=

0.000000	1.00000	2.00000	3.00000	4.00000	5.00000	6.00000	7.00000	8.00000	9.00000	10.0000	9.00000	8.00000	7.00000	6.00000	5.00000	4.00000	3.00000	2.00000	1.00000
1.00000	1.41421	2.23607	3.16228	4.12311	5.09902	6.08276	7.07107	8.06226	9.05539	10.0499	9.05539	8.06226	7.07107	6.08276	5.09902	4.12311	3.16228	2.23607	1.41421
2.00000	2.23607	2.82843	3.60555	4.47214	5.38516	6.32456	7.28011	8.24621	9.21954	10.1980	9.21954	8.24621	7.28011	6.32456	5.38516	4.47214	3.60555	2.82843	2.23607
3.00000	3.16228	3.60555	4.24264	5.00000	5.83095	6.70820	7.61577	8.54400	9.48683	10.4403	9.48683	8.54400	7.61577	6.70820	5.83095	5.00000	4.24264	3.60555	3.16228
4.00000	4.12311	4.47214	5.00000	5.65685	6.40312	7.21110	8.06226	8.94427	9.84886	10.7703	9.84886	8.94427	8.06226	7.21110	6.40312	5.65685	5.00000	4.47214	4.12311
5.00000	5.09902	5.38516	5.83095	6.40312	7.07107	7.81025	8.60233	9.43398	10.2956	11.1803	10.2956	9.43398	8.60233	7.81025	7.07107	6.40312	5.83095	5.38516	5.09902
6.00000	6.08276	6.32456	6.70820	7.21110	7.81025	8.48528	9.21954	10.0000	10.8167	11.6619	10.8167	10.0000	9.21954	8.48528	7.81025	7.21110	6.70820	6.32456	6.08276
7.00000	7.07107	7.28011	7.61577	8.06226	8.60233	9.21954	9.89950	10.6301	11.4018	12.2066	11.4018	10.6301	9.89950	9.21954	8.60233	8.06226	7.61577	7.28011	7.07107
8.00000	8.06226	8.24621	8.54400	8.94427	9.43398	10.0000	10.6301	11.3137	12.0416	12.8062	12.0416	11.3137	10.6301	10.0000	9.43398	8.94427	8.54400	8.24621	8.06226
9.00000	9.05539	9.21954	9.48683	9.84886	10.2956	10.8167	11.4018	12.0416	12.7279	13.4536	12.7279	12.0416	11.4018	10.8167	10.2956	9.84886	9.48683	9.21954	9.05539
10.0000	10.0499	10.1980	10.4403	10.7703	11.1803	11.6619	12.2066	12.8062	13.4536	14.1421	13.4536	12.8062	12.2066	11.6619	11.1803	10.7703	10.4403	10.1980	10.0499
9.00000	9.05539	9.21954	9.48683	9.84886	10.2956	10.8167	11.4018	12.0416	12.7279	13.4536	12.7279	12.0416	11.4018	10.8167	10.2956	9.84886	9.48683	9.21954	9.05539
8.00000	8.06226	8.24621	8.54400	8.94427	9.43398	10.0000	10.6301	11.3137	12.0416	12.8062	12.0416	11.3137	10.6301	10.0000	9.43398	8.94427	8.54400	8.24621	8.06226
7.00000	7.07107	7.28011	7.61577	8.06226	8.60233	9.21954	9.89950	10.6301	11.4018	12.2066	11.4018	10.6301	9.89950	9.21954	8.60233	8.06226	7.61577	7.28011	7.07107
6.00000	6.08276	6.32456	6.70820	7.21110	7.81025	8.48528	9.21954	10.0000	10.8167	11.6619	10.8167	10.0000	9.21954	8.48528	7.81025	7.21110	6.70820	6.32456	6.08276
5.00000	5.09902	5.38516	5.83095	6.40312	7.07107	7.81025	8.60233	9.43398	10.2956	11.1803	10.2956	9.43398	8.60233	7.81025	7.07107	6.40312	5.83095	5.38516	5.09902
4.00000	4.12311	4.47214	5.00000	5.65685	6.40312	7.21110	8.06226	8.94427	9.84886	10.7703	9.84886	8.94427	8.06226	7.21110	6.40312	5.65685	5.00000	4.47214	4.12311
3.00000	3.16228	3.60555	4.24264	5.00000	5.83095	6.70820	7.61577	8.54400	9.48683	10.4403	9.48683	8.54400	7.61577	6.70820	5.83095	5.00000	4.24264	3.60555	3.16228
2.00000	2.23607	2.82843	3.60555	4.47214	5.38516	6.32456	7.28011	8.24621	9.21954	10.1980	9.21954	8.24621	7.28011	6.32456	5.38516	4.47214	3.60555	2.82843	2.23607
1.00000	1.41421	2.23607	3.16228	4.12311	5.09902	6.08276	7.07107	8.06226	9.05539	10.0499	9.05539	8.06226	7.07107	6.08276	5.09902	4.12311	3.16228	2.23607	1.41421

SURFACE,Z

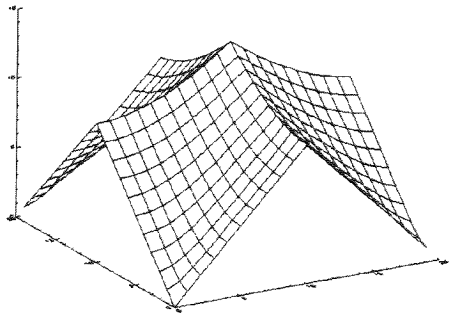


Abbildung 3.2: HTML: Tabelle und Bild

3.2 HTML-Datei-Operationen

Einige der Basisbefehle wurden in größeren Programmen zusammengefasst.

3.2.1 export_html

Mit dieser Routine werden die Operationen `html_file_array`, `html_file_ascii` oder `html_file_struct` zusammengefasst. Diese Prozedur erzeugt von einer IDL-Variable die HTML-Datei `idl_export.html` im TEMP Verzeichnis bei Windows PCs bzw. im $HOME Verzeichnis bei unix Rechnern.

html_file_array

Diese Routine kombiniert `html_header`, `array2htmltable` und `html_trailer` Funktionen und schreibt eine Datei.

html_file_ascii

Diese Routine kombiniert `html_header`, `ascii2html` und `html_trailer` Funktionen und schreibt eine Datei.

html_file_struct

Diese Routine kombiniert `html_header`, `struct2htmltable` und `html_trailer` Funktionen und schreibt eine Datei.

3.3 HTML-Verzeichnis-Operationen

Die folgenden Routinen ermöglichen es einen Verzeichnisbaum in HTML darzustellen.

3.3.1 dir2htmlfile()

Diese Funktion schreibt Informationen von einem Verzeichnis in eine HTML-Datei.

3.3.2 tree2htmlfile()

Diese Funktion erzeugt eine HTML-Datei von einem Verzeichnisbaum in Form einer Tabelle.

3.3.3 dirtree2htmlfile()

Diese Funktion erzeugt eine HTML-Datei von einem Verzeichnisbaum in Form eines Explorer Verzeichnisbaums.
Als Basisfunktion wird das `javascript foldertree` siehe [*M. A. Martins* (1997)] verwendet.

```
dirtree2htmlfile,'S:/links/daten/experiments/stream1998.html',$
    'S:/links/daten/experiments/stream1998',level=10
```

Beispiel 3.12: HTML: dirtree2htmlfile

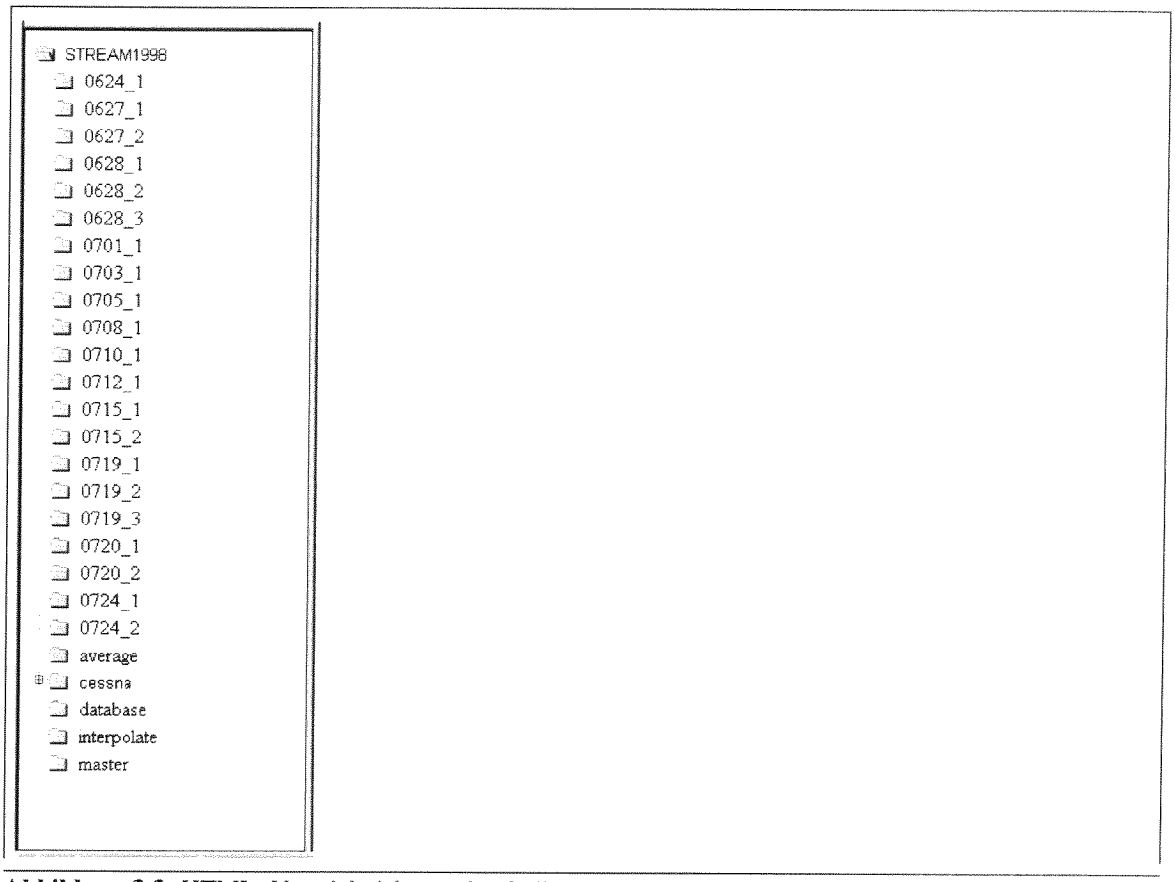


Abbildung 3.3: HTML: Verzeichnisbaum durch dirtree2htmlfile

3.3.3 dirtree2htmlfile()

Diese Funktion erzeugt eine HTML-Datei von einem Verzeichnisbaum in Form eines Explorer Verzeichnisbaums.
Als Basisfunktion wird das `javascript foldertree` siehe [*M. A. Martins* (1997)] verwendet.

```
dirtree2htmlfile,'S:/links/daten/experiments/stream1998.html',$
    'S:/links/daten/experiments/stream1998',level=10
```

Beispiel 3.12: HTML: `dirtree2htmlfile`

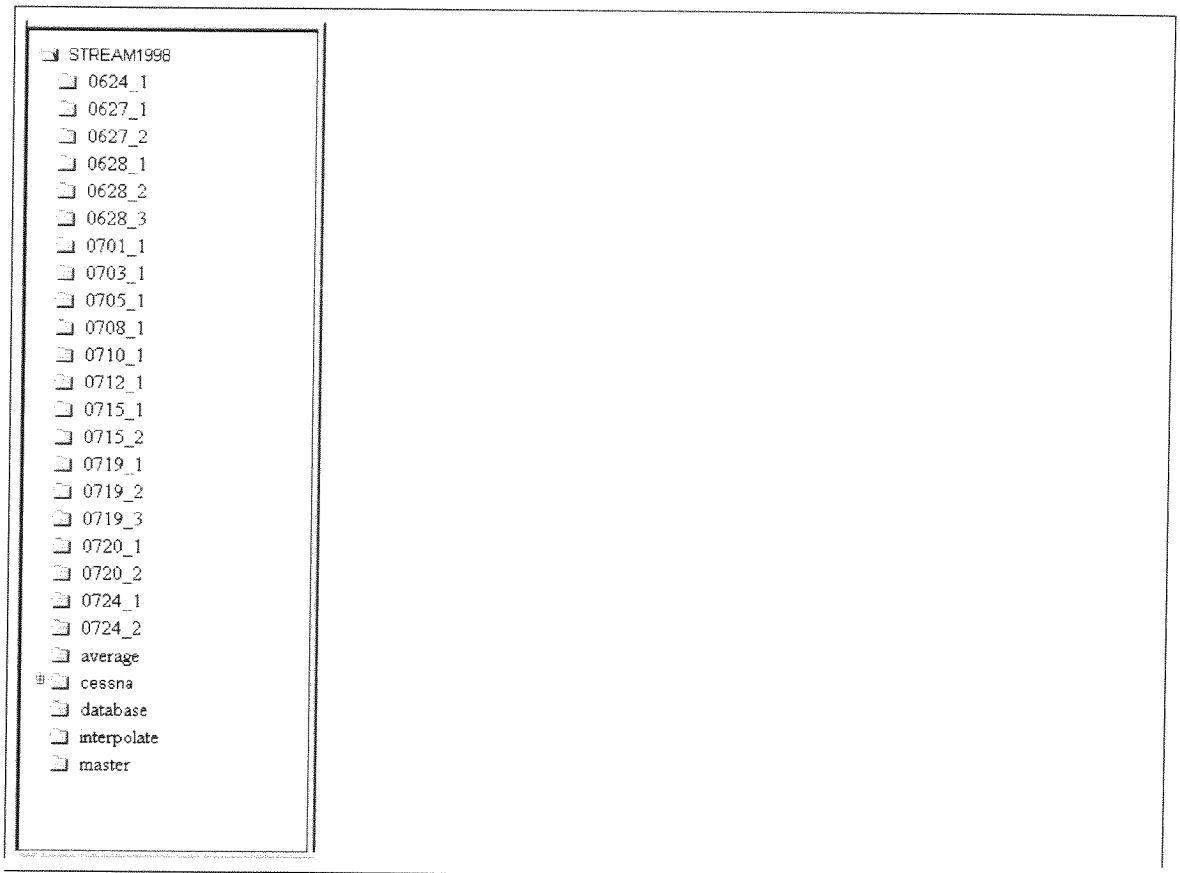


Abbildung 3.3: HTML: Verzeichnisbaum durch dirtree2htmlfile

3.3.3 dirtree2htmlfile()

Diese Funktion erzeugt eine HTML-Datei von einem Verzeichnisbaum in Form eines Explorer Verzeichnisbaums.
Als Basisfunktion wird das `javascript foldertree` siehe [*M. A. Martins* (1997)] verwendet.

```
dirtree2htmlfile,'S:/links/daten/experiments/stream1998.html',$
    'S:/links/daten/experiments/stream1998',level=10
```

Beispiel 3.12: HTML: dirtree2htmlfile

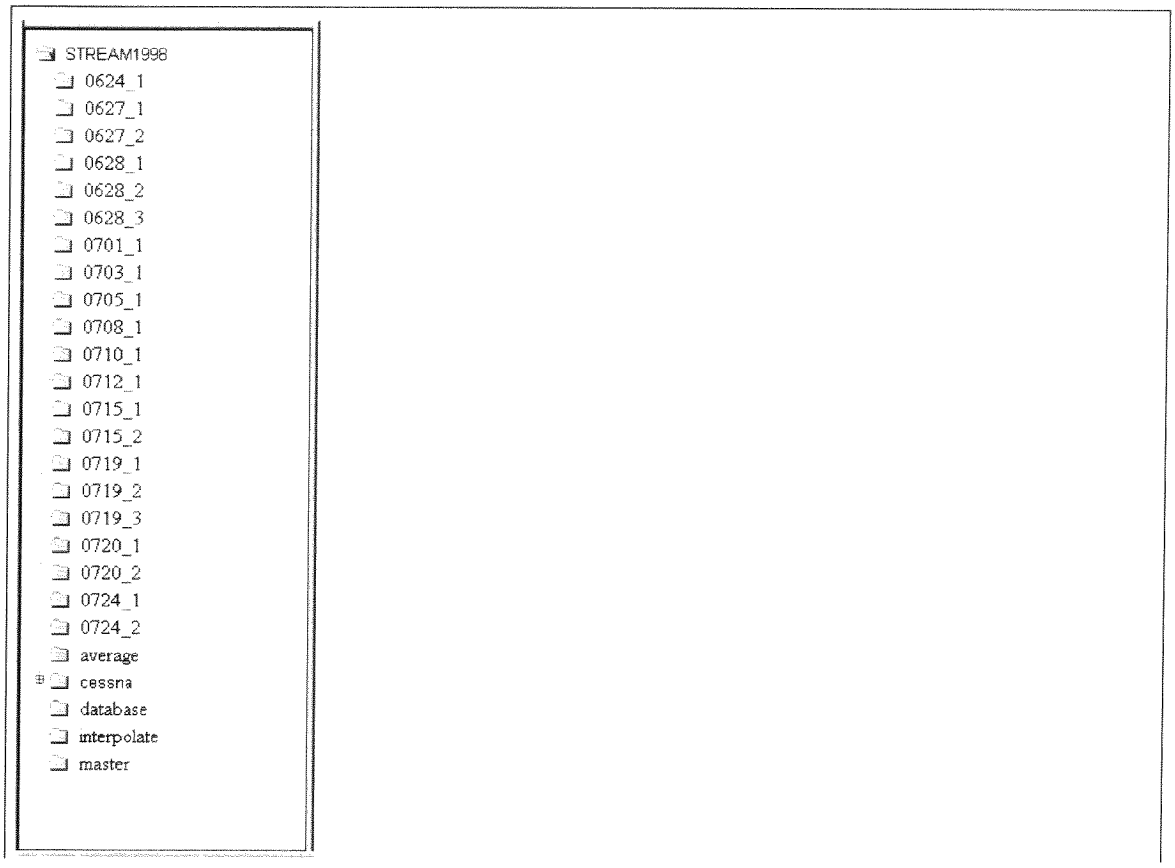


Abbildung 3.3: HTML: Verzeichnisbaum durch dirtree2htmlfile

Kapitel 4

Die ICG-Daten-Struktur

Die ICG-Daten-Struktur ist eine Definition zur Organisation von Daten. Sie wurde im August/September 1998 auf Grund langer Diskussionen mit Anwendern im ICG definiert.
Diese Datenstruktur sollte folgende Bedingungen erfüllen:

- sie muss zum netCDF Daten-Format, siehe [*R. Rew et al.*(1992)] kompatibel sein. Dadurch wird ein externer Standard benutzt und Datenaustausch ermöglicht, siehe Beispiel 4.3.1 auf Seite 50.
- beliebige Dimensionierung von Datenfeldern in Abhängigkeit einer beliebigen Koordinatenvariablen, siehe Beispiel 4.1.6 auf Seite 38
- Darstellbarkeit mehrerer Parameter, siehe Beispiel 4.1.6 auf Seite 34
- möglichst geringe Eingabewege, (am Besten nur *struct.clo*), siehe Beispiel 4.2.3 auf Seite 45
- Erleichterung des Aufbaus einer Datenbank, siehe Kapitel 6.2.2 auf Seite 132.

Im Dezember 1998 wurde der Entwurf in der IDL_SOURCE-Arbeitsgruppe vorgestellt. Nach kleinen Änderungen wurde diese Daten-Struktur als Arbeitsgrundlage für zukünftige Projekte definiert. Seit diesem Zeitpunkt hat sie auch den Namen ICG-Daten-Struktur. Die gebräuchlichsten Abkürzungen sind icgs und icg_struct.

4.1 Definition

Die Einträge der ICG-Daten-Struktur basieren im wesentlichen auf Definitionen des international gängigen Dateiformats netCDF, das von der Firma unidata definiert wurde, siehe [*R. Rew et al.*(1992)].
Sie stellt damit ein Abbild einer netCDF-Datei dar. Neben den reinen Daten kann sie auch alle Randbedingungen enthalten, die zusammen mit den Daten den Datensatz definieren. Demzufolge wird eine Unterscheidung in globale Attribute, Parameter und Parameter Attribute vorgenommen.
Auf der ersten Ebene enthält eine ICG-Daten-Struktur zwei Gruppen von Einträgen: Parameternamen und sonstige Informationen. Die sonstigen Informationen beginnen alle mit einem Ausrufezeichen.
Die globalen Attribute (siehe Definition 4.1.1 auf Seite 27) werden alle unter *!GLOBAL* in einzelnen Unter-Strukturen zusammen gefasst. Die Einträge werden für *.PI*, *.DATASET*, *.PLATFORM*, *.EXP* und *.HISTORY* unterschieden.
Wird die Struktur aus einer Datei erzeugt, enthält sie auf dem *!FILENAME* den verwendeten Dateinamen.
Das Zeichen *!* ist für die weitere Entwicklung der Struktur reserviert.
Alle nicht mit *!* beginnenden Struktureinträge sind Parameter-Strukturen. Innerhalb dieser Strukturen sind auch die Parameter-spezifischen Attribute (siehe Definition 4.1.2 auf Seite 29) eingetragen.

Ein besonderes Parameter Attribut ist *short_name*. Der darin eingetragene Name ist der Parameter-Name, mit dem der Parameter in der netCDF-Datei abgebildet wird. Das bedeutet, dass man den Tag-Namen, des Parameters (in Abbildung, siehe 4.1 ist er als VALUE benannt) innerhalb der ICG-Daten-Struktur frei wählen kann.
Die Werte der Parameter-Struktur von 'time' stellen die Mitte des Messintervalls dar. Die Einheit der Zeit ist 'seconds since 2000-01-01 00:00:00 UTC'. Dieses Zeitformat wurde von Ray Sterner (siehe [*R. Sterner et al.*(1992)]) definiert und wird auch *'julian seconds'* genannt.
Um die Parameter-Definition in Bezug auf Dimensions-Größe und Dimensions-Name frei wählen zu können, kann man unter jeder Parameter-Struktur eine *STATUS-Struktur* (siehe Kapitel 4.1.4 auf Seite 32) einbauen. Dieser Mechanismus ist dann wichtig, wenn man nicht nur Vektoren, sondern mehrdimensionale Felder oder unterschiedliche Dimensionierungen in einer Datei abbilden will.
Eine kleine Zahl der möglichen Attribute sollte immer gesetzt werden. Durch diese Attribute ist eine eindeutige Zuordnung der Daten und ihre Bedeutung gesichert. Diese Struktur ermöglicht es spezifische Eigenschaften der Daten, wie z.B.

- wer der Verantworliche ist (PI),
- wo gemessen wurde (PLATFORM),
- zu welcher Kampagne die Messung gehört (DATASET),
- zu welcher Kategorie die Daten gehören (DATASET),
- welches Experiment die Daten gesammelt hat (DATASET),
- was der Zweck der Daten ist (DATASET),
- um welche Daten es sich handelt (DATASET),
- welcher Achsentitel für den Parameter verwendet wird (VALUE),
- welche Einheit der Parameter hat (VALUE),
- für welchen Zeitbereich der Parameter gilt (VALUE),

mit den Daten zusammen zu speichern. Mit der Abbildung 4.1 wird der Aufbau der Struktur dargestellt.
Mit der Funktion `q_icg_struct()` (siehe Kapitel 4.2.2 auf Seite 43) kann man die Struktur auf diese Kontinuität prüfen. Dieser Test wird auch in `write_ncdf` (siehe Kapitel 4.3.1 auf Seite 50) vorgenommen.

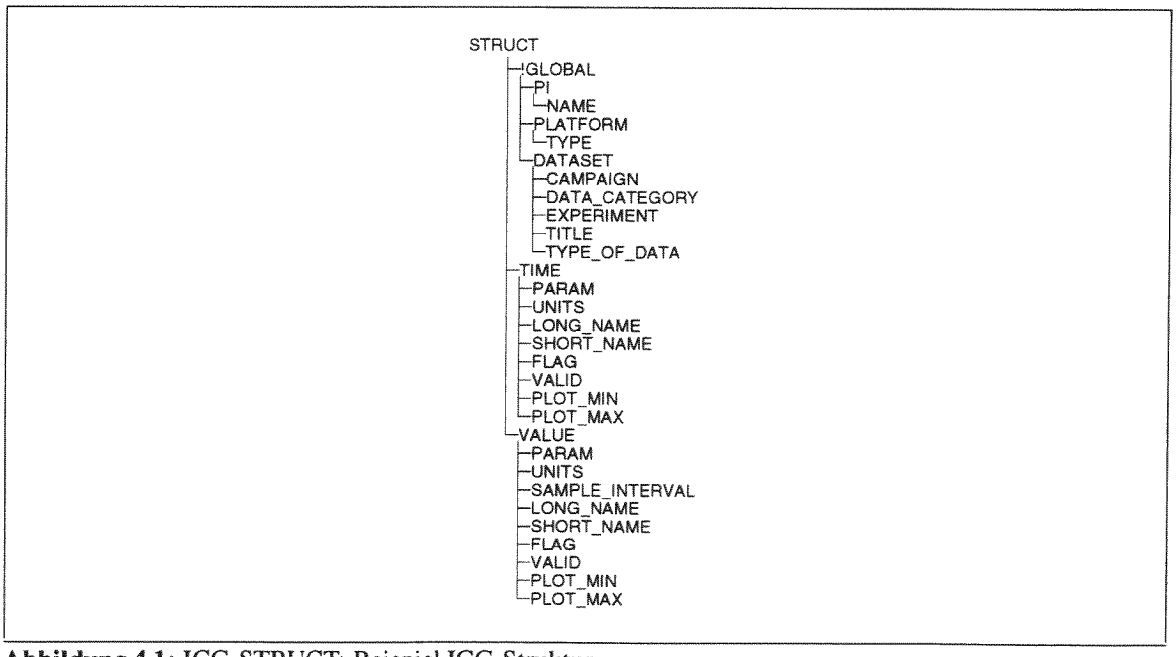


Abbildung 4.1: ICG_STRUCT: Beispiel ICG-Struktur

4.1.1 Globale Attribute

Die globalen Attribute in Abbildung 4.1 auf der vorherigen Seite sind unterhalb der Struktur *.!GLOBAL* in den Strukturen *.PI*, *.DATASET*, *.PLATFORM* und den Variablen *.COMMENT* und *.HISTORY* eingeordnet. In der folgenden Tabelle sind die hier möglichen Einträge aufgeführt.

Die unter **I** mit X gekennzeichneten Elemente müssen immer definiert sein. Für die Verarbeitung bzw. den Austausch der Daten mit anderen Gruppen ist es wünschenswert, daß auch alle (X) Elemente definiert sind.

Die mit XW gekennzeichneten Attribute *creation_time* und *modification_time* werden durch `write_ncdf` automatisch im Format: YYYY-MM-DD hh:mm:ss gesetzt.

Variable:	**Beschreibung:**	**I**	**Länge**
COMMENT	Kommentar		CHAR 32767
HISTORY	Entwicklung der Daten Trennz. !C	X	CHAR 32767

Tabelle 4.1: ICG_STRUCT: Definition: Globale Attribute: !GLOBAL.

Struktur:	**Attribut Namen:**	**Beschreibung:**	**I**	**Länge**
PI	name	Ansprechpartner	X	CHAR 512
	workgroup	Arbeitsgruppe		CHAR 512
	organisation	Organisation	(X)	CHAR 512
	institute	Institut		CHAR 512
	address	Adresse		CHAR 512
	phone_number	Telefonnummer		CHAR 512
	email	E-Mail Adresse des PI		CHAR 512

Tabelle 4.2: ICG_STRUCT: Definition: Globale Attribute: !GLOBAL.PI.

Struktur:	Attribut Namen:	Beschreibung:	I	Länge
DATASET	title	engl. Titel der Datentabelle	X	CHAR 512
	de_title	Titel der Datentabelle		CHAR 512
	data_category	'EXPERIMENT' oder 'MODEL '	X	CHAR 512
	data_description	Typ der Daten die von unter data_origin gespeicherten Institution geholt wurden (z.B. UKMO, ECMWF)		CHAR 512
	data_forecast_hours	Bei Vorhersage-Daten wird hier die Vorhersagezeit in Stunden vermerkt, damit diese von den Analysedaten unterschieden werden können.		LONG
	experiment	bei Experimenten, 'FISH' bei den Modellen, Name des Programms,mit dem der Datensatz erzeugt wurde	X	CHAR 512
	data_origin	z.B. RAW, MID, NASA		CHAR 512
	campaign	Name der Kampagne, z.B.: STREAM1998	(X)	CHAR 512
	mission	Kennzeichen der Mission	(X)	CHAR 512
	type_of_data	bei Experimenten: 'MEASUREMTS' od. 'CALIBRATION' bei Modellen: 'CALCULATION'	X	CHAR 512
	status_of_file	Status der Daten z.B. prelimenary	(X)	CHAR 512
	version	Version der Datei z.B. 1		CHAR 512
	source_file_name	Name der Ursprungs Datei		CHAR 512
	creation_time	Erstellungsdatum der Datei	XW	CHAR 512
	modification_time	Änderungsdatum der Datei	XW	CHAR 512
	languages	Sprachinformationen		CHAR 512

Tabelle 4.3: ICG_STRUCT: Definition: Globale Attribute: !GLOBAL.DATASET.

Struktur:	Attribut Namen:	Beschreibung:	I	Länge
PLATFORM	type	Typ der Plattform, z.B.: 'AIRCRAFT' bei Modellen: 'COMPUTER'	X	CHAR 512
	name	Eigenname der Plattform	(X)	CHAR 512

Tabelle 4.4: ICG_STRUCT: Definition: Globale Attribute: !GLOBAL.PLATFORM.

Weitere globale Attribute sind die experimentspezifischen Attribute; diese sind unterhalb des Struktursammel-Punktes *.EXP* eingeordnet. Die Verwendung der *EXP-Struktur* erlaubt es dem Benutzer, globale Definitionen selbst zu treffen. Die Verwendung geht aus den Beispielen hervor (siehe Beispiel 4.1.6 auf Seite 36). Es ist eine bestimmte Syntax nötig, damit eine Übertragung der Attribute in andere Dateien möglich ist.
Das globale Attribut *!FILENAME* wird verwendet, um den Namen der Datei anzugeben, deren Daten in der ICG-Daten-Struktur abgebildet wurden.

4.1.2 Parameter-Attribute

Die Parameter-Attribute sind unterhalb jeder Parameter-Struktur (siehe TIME und VALUE in Abbildung 4.1 auf Seite 26) eingetragen. In der Tabelle 4.5 werden die möglichen Eintragungen beschrieben. Die unter **I** mit X gekennzeichneten Elemente müssen immer definiert sein. Das sample_interval muß nicht gesetzt werden, wenn man entweder *INTERVAL_T* oder *BEGIN_T* und *END_T* als *flag* (siehe Definition 4.1.3 auf Seite 31) definiert hat.

Attribut Namen:	**Beschreibung:**	**I**	**Länge**
sample_interval	gültiger Zeitbereich für die Werte	(X)	DOUBLE
description	engl. Beschreibung des Parameters		CHAR 512
de_description	dt. Beschreibung des Parameters		CHAR 512
scale_factor	Skalierungsfaktor des Parameters		DOUBLE
add_offset	Offset		DOUBLE
units	Einheit des Parameters	X	CHAR 512
long_name	engl. Achsenbeschriftung	X	CHAR 512
de_long_name	dt. Achsenbeschriftung		CHAR 512
FORTRAN_format	Format der Zahlen im Fortran Format		CHAR 512
C_format	Format der Zahlen in C		CHAR 512
valid_min	untere Grenze gültiger Werte		pt
valid_max	obere Grenze gültiger Werte		pt
detection_limit	die gleiche Nachweißgrenze für alle Werte		pt
stdev_abs	Standardabweichung absolut für alle Werte		pt
stdev_rel	Standardabweichung relativ für alle Werte		pt
no_of_points	Anzahl der Punkte für alle Werte		LONG
accuracy_abs	absolute Genauigkeit für alle Werte		pt
accuracy_rel	relative Genauigkeit für alle Werte		pt
missing_value	gemessen aber kl. detection_limit		pt
fill_value	keine Messung, daher Füll-Variable		pt
param	Daten	X	
short_name	Bezeichner der Daten in einer Datei	X	CHAR 512
status	Struktur mit zus. netCDF Eigenschaften		CHAR 512
flag	beschr. Vektoren, die zu dem Parameter gehören, default: NONE	X	CHAR 512

pt bedeutet Länge wie param

Tabelle 4.5: ICG_STRUCT: Definition: Parameter Attribute

Der Daten-Typ von *param* ist frei wählbar. Dadurch ist eine Speicherplatz schonende Speicherung von Daten möglich. Der Daten-Typ STRING für den Parameter wird als BYTE-ARRAY verarbeitet. Dadurch haben STRING-Daten zwei Dimensionen.

Wenn in einer netCDF Datei das Attribut *add_offset* oder/und das Attribut *scale_factor* gesetzt ist, werden deren Werte mit den Werten des Parameters und den entsprechenden flags verrechnet. Danach wird das *add_offset* Attribut in der Struktur auf 0.D und das *scale_factor* Attribut auf 1.D gesetzt. Die Rechenvorschrift lautet: X = X * scale_factor + add_offset. Zusätzlich wird dann auf den Attributen *add_offset_previous* bzw. *scale_factor_previous* der ursprügliche Wert angezeigt.

Die nachfolgenden Parameter-Attribute unterhalb jeder Parameter-Struktur werden dynamisch für jeden Parameter erzeugt. Dazu werden die Funktionen `def_valid_for_param()` , `icg_struct_param_valid()` , `valid_param()` oder `update_valid_index` verwendet (siehe Kapitel 4.1.5 auf Seite 33).

Attribut Namen:	**Beschreibung:**	**I**	**Länge**
valid	die indizes der gültigen Werte in Abhängigkeit zu valid_min, valid_max, fill_value, missing_value und dem flag QUALITY, siehe Tabelle 4.7 auf der nächsten Seite		LONG
missed	sofern missing_value gesetzt ist enthält missed ein array mit den indizes der durch missing_value fehlenden Werte		LONG
filled	sofern fill_value gesetzt ist enthält filled ein array mit den indizes der durch fill_value fehlenden Werte		LONG
plot_min	enthält min(param[valid])		pt
plot_max	enthält max(param[valid])		pt

pt bedeutet Länge wie param

Bei Modelldaten sind in der Regel alle Daten gültig. Daher ist die Anlegung eines valid-Feldes ungünstig da es dann häufig die gleiche Anzahl Elemente, wie der Parameter enthält. Für diese Daten ist die Darstellung von not_valid günstiger. Daraus ergeben sich nachfolgende Eintragungen in der ICG-Daten-Struktur.

Attribut Namen:	**Beschreibung:**	**I**	**Länge**
not_valid	die indizes der ungültigen Werte in Abhängigkeit zu valid_min, valid_max, fill_value, missing_value und dem flag QUALITY, siehe Tabelle 4.7 auf der nächsten Seite		LONG
plot_min	enthält min(param[valid])		pt
plot_max	enthält max(param[valid])		pt

pt bedeutet Länge wie param

Tabelle 4.6: ICG_STRUCT: Definition: dynamisch erzeugte Parameter Attribute

4.1.3 flag

Durch Definition des Parameter-Attributs *flag* können weitere Parameter Attribute in der gleichen Dimensionierung wie der korrespondierende Parameter definiert werden. Das Trennzeichen der einzelnen flag-Attribute ist ein Leerzeichen. Mit dem Eintrag NONE wird gekennzeichnet, dass kein flag beim Speichern in eine Datei berücksichtigt werden muß (Voreinstellung). Die flags beinhalten die gleiche Dimensionierung wie der Bezugs-Parameter. Die flag-Bezeichner müssen in Grossbuchstaben geschrieben werden. In der netCDF Datei werden sie mit dem Namen *short_name@flag* abgelegt, z.B.: *Br@DETECTION_LIMIT*

Für Parameter (außer time) gibt es für das Parameter Attribut flag folgende Möglichkeiten:

flag-Name	**Beschreibung:**
ACCURACY	Genauigkeit der Werte
DETECTION_LIMIT	Nachweisgrenze
DISTANCE	0 bedeutet keine Interpolation, da eine genaue Zeitübereinstimmung vorlag. nn in Sekunden entspricht dem Abstand der beiden Interpolationsstützpunkte.
EST_NEG	Abgeschätzter negativer Fehler
EST_POS	Abgeschätzter positiver Fehler
EXT_ERR	Fehler des Mittelwertes aus der Stichprobenstatistik
LOW_ERR	Unterer Fehler
MIN	Minimum der gemittelten Werte
MAX	Maximum der gemittelten Werte
HIGH_ERR	Oberer Fehler
INT_ERR	der interne Fehler, z.B. von einem Mittelwert
MEDIAN	der Median der gemittelten Werte
MODE	Messmodus
NO_OF_POINTS	Anzahl der Werte, die in die Mittelung eingegangen sind
STDEV	Standardabweichung der gemittelten Werte
QUALITY	0 = gut 1 = unbekannter Fehler 2 = Meßbereich unterschritten 4 = Meßbereich überschritten 8 = Außerhalb des Bereichs 16 = Unterhalb der Nachweisgrenze 32 = Benutzereingriff 64 = Unbekanntes Problem Weitere Merkmale, kann der Benutzer selber frei definieren und beschreiben.

Tabelle 4.7: ICG_STRUCT: Definition: FLAG Parameter für Parameter (außer der Zeit)

Für den Parameter time gibt es folgende Möglichkeiten:

flag-Name:	**Beschreibung:**
BEGIN_T	Anfangszeit des Mittelungsfensters
END_T	Endzeit des Mittelungsfensters
INTERVAL_T	Das Messintervall

Tabelle 4.8: ICG_STRUCT: Definition: FLAG Parameter für die Zeit

4.1.4 STATUS-Struktur

Wenn ein Datenpunkt genau einem anderen Datenpunkt zugeordnet ist, wird dies als 1:1 Beziehung bezeichnet. Ist er mehreren anderen Datenpunkten zugeordnet liegt eine 1:n Beziehung vor. Viele unserer Messdaten liegen als 1:1 Beziehungen in Form von Vektoren in der Abhängigkeit zu einem Vektor Zeit vor. Die Dimension der Zeit bestimmt somit die Anzahl der Werte in den Messdaten-Vektoren. Daher ist der Parameter *time* gleichzeitig auch die Dimensionsvariable. In diesem Fall wird *time* als Koordinatenvariable bezeichnet.

Falls weitere oder andere Abhängigkeiten vorhanden sind, müssen diese mit Hilfe der STATUS-Struktur in der ICG-Daten-Struktur definiert werden z.B. bei Satelliten oder Modell-Daten. Bei diesen sind häufig Druck, Temperatur u.a. als weitere Koordinatenvariablen oder Dimensionsvariablen zu berücksichtigen.

Die Attribute werden durch */STATUS* beim Lesen aus einer netCDF-Datei mit `read_ncdf()` belegt. Beim Definieren einer ICG-Daten-Struktur kann man folgende Attribute setzen.

Struktur:	**Attribut Namen:**	**Beschreibung:**	**Länge**
param	creator	Benutzername, der den Parameter angelegt hat	CHAR 512
	creation_time	Zeitpunkt des ersten Parameter schreibens	CHAR 512
	modification_time	Zeitpunkt der Modification des Parameter	CHAR 512
	type	Typ der Werte des Parameters	INT

Tabelle 4.9: ICG_STRUCT: Definition: PARAM Unterstruktur von der STATUS Struktur

Die Attribute *creation_time* und *modification_time* werden durch `write_ncdf` automatisch im Format: YYYY-MM-DD hh:mm:ss gesetzt. Der Eintrag von *creator* entspricht dem Benutzernamen. Mit dem Eintrag auf type kann der Wertetyp des Parameters (unabhängig der Werte) festgelegt werden.

Angelegte Variablen, deren *short_name* auch der *status.dim.name* ist, werden Koordinaten-Variablen genannt. In netCDF werden Koordinaten-Variablen in Kleinbuchstaben definiert. Bei mehr-dimensionalen Feldern oder wenn man Koordinaten-Variablen definieren möchte, muss man die *status.dim-Struktur* verwenden, (siehe Beispiel: 4.1.6 auf Seite 38). *.size* muss man nur dann verwenden, wenn man größere Felder vorinitialisieren möchte. Verwendet man selbst *.size* ist es wichtig, dass die Dimensionierung auf keinem Fall kleiner als die Größe des Parameters ist. Das netCDF verlangt einen Dimensions-Namen für eine Dimensionierung in *.name*. Nur bei Daten, die nur von *time* abghängig sind, ist es nicht nötig, den Dimensions-Namen *'time'* zu setzen.

Struktur:	**Attribut Namen:**	**Beschreibung:**	**Länge**
dim	name	Skalar oder Vektor mit Dimensions Namen	CHAR 512
	size	Skalar oder Vektor mit Dimensions Längen	LONG

Tabelle 4.10: ICG_STRUCT: Definition: DIM Unterstruktur von der STATUS Struktur

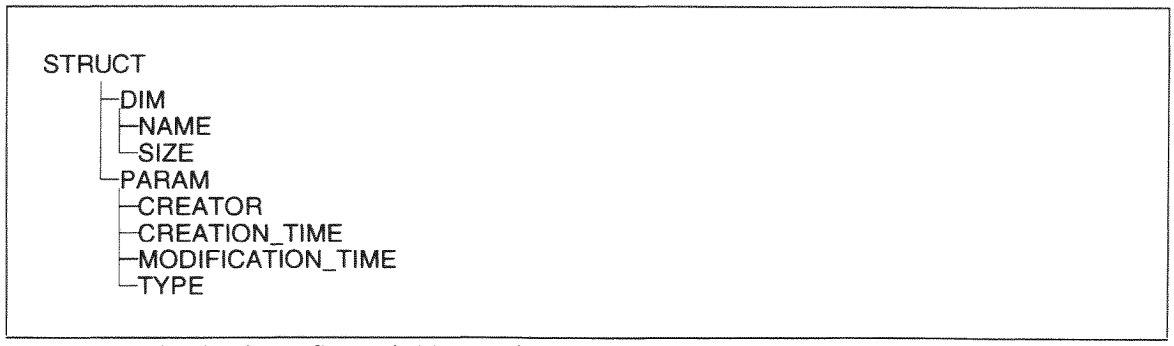


Abbildung 4.2: ICG_STRUCT: Definition: Aufbau der STATUS Struktur

4.1.5 Valid-Struktur

Die Tag-Namen der Valid-Struktur befinden sich in den Parameter-Strukturen. Sie werden durch die Funktion `valid_param` (siehe Kapitel 4.2.5 auf Seite 50) erzeugt. Diese Informationen werden nicht in netCDF-Dateien gespeichert.

Attribut Namen:	Beschreibung:
valid	enthält die Indizes der gültigen Werte, ohne missing_value und ohne fill_value und falls der flag QUALITY gesetzt ist nur Werte bei denen QUALITY=0 ist.
missed	enthält die Indizes die durch das missing_value gekennzeichnet sind.
filled	enthält die Indizes die durch das fill_value gekennzeichnet sind.
plot_min	enthält den Minimum-Wert der gültigen Werte.
plot_max	enthält den Maximum-Wert der gültigen Werte.

Tabelle 4.11: ICG_STRUCT: Definition: Valid Struktur

Das Steuerwort `/not_valid` ergibt anstelle des Valid-Indexfeldes dessen Umkehrung in dem not_valid-Indexfeld. Die Felder missed und filled entfallen mit dieser Option.

4.1.6 Erstellung der Datenstruktur

Um den Einstieg zu erleichtern, werden an dieser Stelle einige ausgewählte Beispiele definiert. In den Beispielen werden folgende Definitionen gezeigt:

1. typische Experimente Daten; Verwendung des flag Attributes für die Angabe der Standardabweichung; alle Variablen sind abhängig von der Zeit, daher wird keine STATUS-Struktur benötigt, siehe Beispiel 4.1 auf der nächsten Seite.
2. Verwendung der EXP-Struktur; Experiment spezifische Attribute werden zum Beispiel 4.1 auf der nächsten Seite im Beispiel 4.2 auf Seite 36 ergänzt.
3. Verwendung der STATUS-Struktur, da die Daten zusätzlich zur Zeit einem weiteren Parameter zugeordnet sind, ist eine Zuweisung einer zusätzlichen Koordinatenvariablen nötig, siehe Beispiel 4.3 auf Seite 38.
4. keine Abhängigkeit zur Zeit, dafür andere Abhängigkeiten; mit der Definition der STATUS-Struktur werden die anderen Abhängigkeiten dargestellt, siehe Beispiel 4.4 auf Seite 40.

Als Hilfe bei der Visualisierung der Struktur kann man `struct_tree` oder `html_file_struct` verwenden. `struct_tree` zeigt den Aufbau der Struktur in Form eines Baumes, siehe Abbildung 4.1 auf Seite 26 und `html_file_struct` die Daten der Struktur in Form von Tabellen, siehe Abbildung 4.1.6 auf Seite 42.

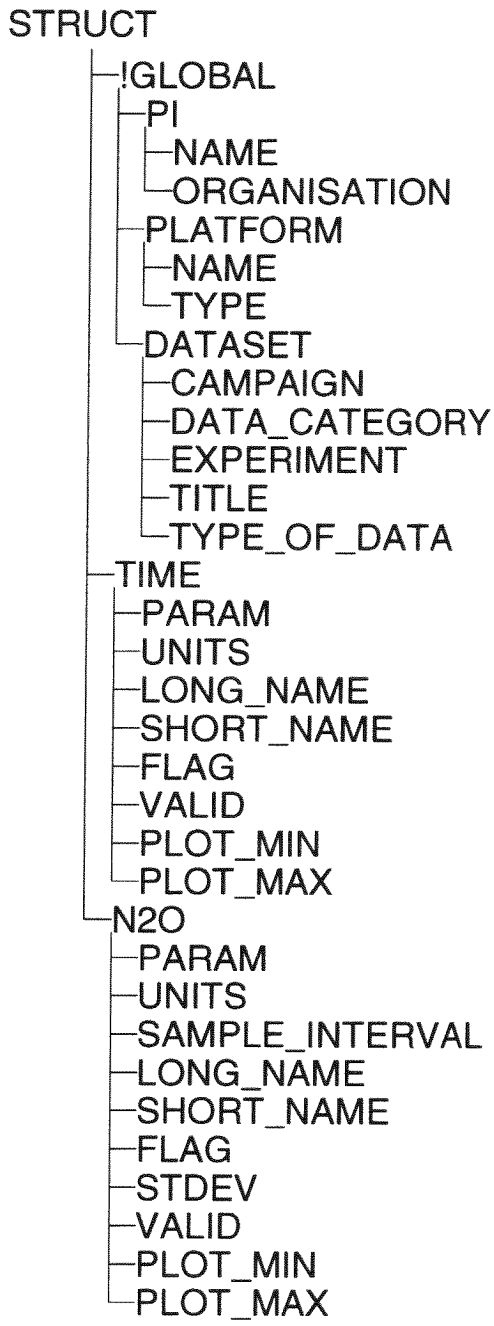
Definition einer Datenstruktur: Alle Variablen abhängig von time

Dieses Beispiel entspricht mit dem Unterschied des verwendeten FLAGs dem Beispiel 4.1 auf Seite 26. Der Parameter N2O ist nur der Zeit in einer 1:1 Beziehung zugeordnet. Daher wird die STATUS-Struktur nicht benötigt.

```
pi={name:'P.Mustermann',organisation:'FZ Jülich'}
platform={name:'Cessna',type:'AIRCRAFT'}
dataset={campaign:'OTTO1999',DATA_CATEGORY:'EXPERIMENT',$
          experiment:'GHOST',title:'validation',$
          type_of_data:'MEASUREMENTS'}
global=CREATE_STRUCT('pi',pi,'platform',platform,'dataset',dataset)
time={param:DINDGEN(100), $
       units:'seconds since 2000-01-01 00:00:00 UTC',$
       long_name:'time',short_name:'time',flag:'NONE'}
time=CREATE_STRUCT(time,valid_param(time))
n2o={param:FINDGEN(100),units:'ppm',$
      sample_interval:1.0,$
      long_name:'N2O',short_name:'N2O',$
      flag:'STDEV',stdev:sin(dindgen(100))}
n2o=CREATE_STRUCT(n2o,valid_param(n2o))

struct=CREATE_STRUCT('!global',global,'time',time,'n2o',n2o)

struct_tree,struct

;html_file_struct,'test.html',struct, /subtab,/border,/fixed, $
     /blockquote,max=2
```

Beispiel 4.1: ICG_STRUCT: Definition: Alle Variablen abhängig von time

Schematischer Aufbau der Struktur aus Beispiel 4.1 auf der vorherigen Seite:



Definition von experimentspezifischen Attributen

Zusätzlich zu der Definition der Struktur von Beispiel 4.1 auf Seite 34, werden experimentspezifische Daten mit der EXP-Struktur aufgenommen.

```
pi={name:'P.Mustermann',organisation:'FZ Jülich'}
platform={name:'cessna',type:'AIRCRAFT'}
dataset={campaign:'OTTO1999',DATA_CATEGORY:'EXPERIMENT',$
          experiment:'GHOST',title:'validation',$
          type_of_data:'MEASUREMENTS'}
; Definition von User Attributen
GHOST={FC1_type:'BROOKS',FC1_last_calibrated:'1999-02-02', $
       FC1_intern_no:4}
exp={ghost:ghost}
history='info_file:add001.txt'
global=CREATE_STRUCT('pi',pi,'platform',platform,$
       'dataset',dataset,'exp',exp,'history',history)
time={param:DINDGEN(100), $
      units:'seconds since 2000-01-01 00:00:00 UTC',$
      long_name:'time',short_name:'time',flag:'NONE'}
time=create_struct(time,valid_param(time))
n2o={param:FINDGEN(100),units:'ppm',$
     sample_interval:1.0,$
     long_name:'N2O',short_name:'N2O',$
     flag:'STDEV',stdev:sin(dindgen(100))}
n2o=create_struct(n2o,valid_param(n2o))
struct=CREATE_STRUCT('!global',global,'time',time,'n2o',n2o)

struct_tree,struct

; html_file_struct,'test.html',struct, /subtab,/border,/fixed,max=10
```

Beispiel 4.2: ICG_STRUCT: Definition von Experiment Spezifischen Attributen

Da das Experiment GHOST heißt, müssen die zusätzlichen, für dieses Experiment spezifischen Attribute in der Struktur GHOST zusammengefasst werden. Dies zeigt das nachfolgende Bild.

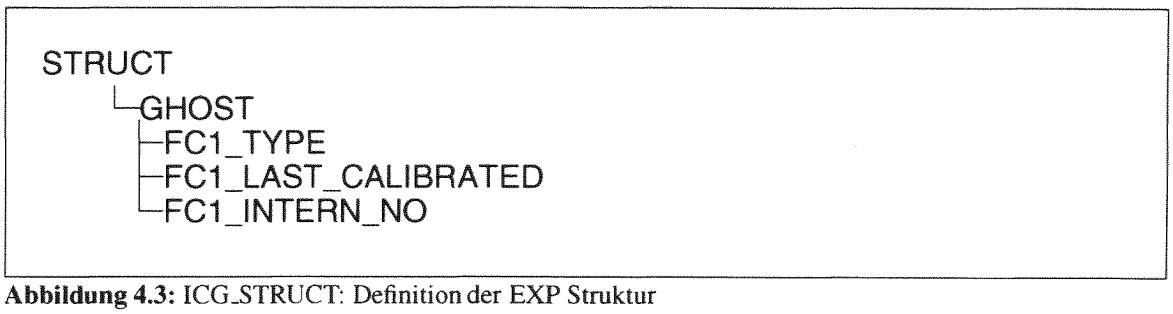
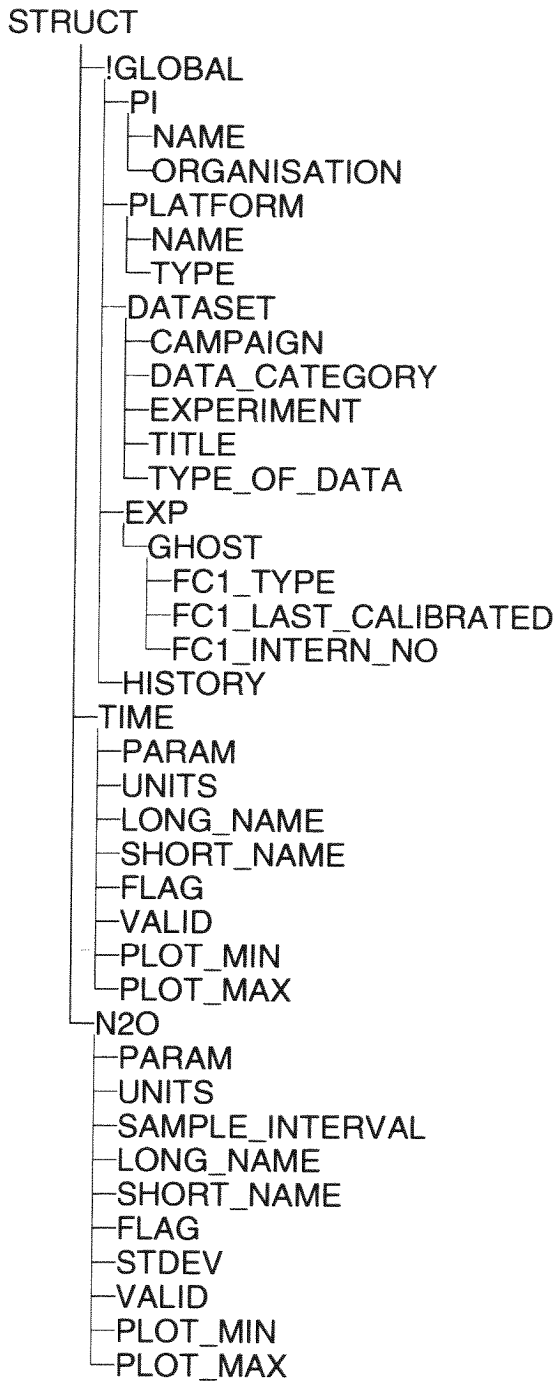


Abbildung 4.3: ICG_STRUCT: Definition der EXP Struktur

Schematischer Aufbau der Struktur aus Beispiel 4.2 auf der vorherigen Seite:



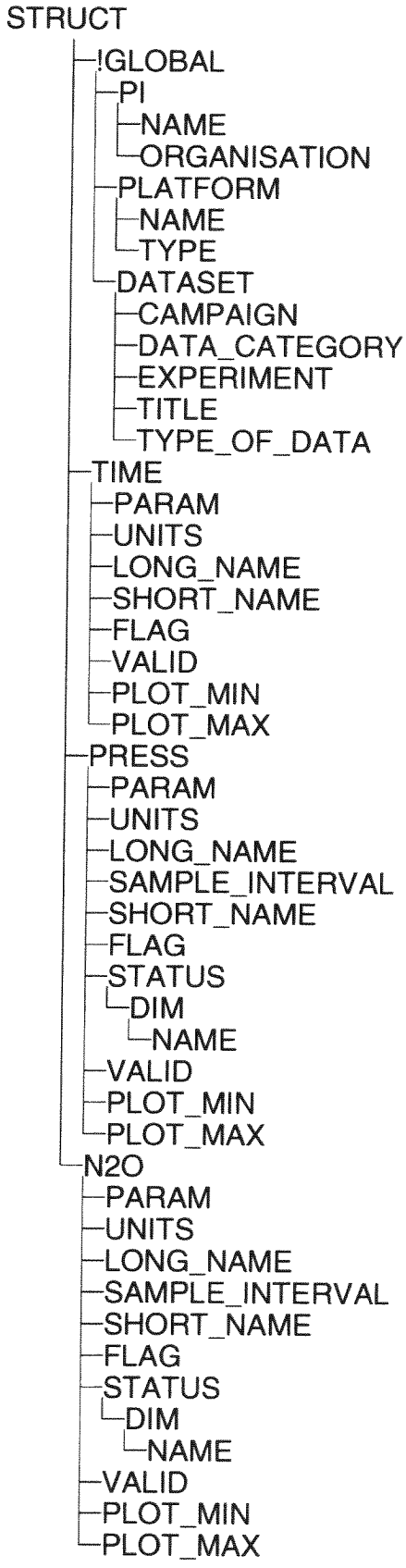
Definition einer Datenstruktur mit einer zusätzlichen Abhängigkeit ausser time

Dieses Beispiel zeigt die Verwendung einer weiteren Koordinatenvariable (PRESS) in der Struktur mit Hilfe der STATUS-Struktur.

```
pi={name:'P.Mustermann',organisation:'FZ Jülich'}
platform={name:'cessna',type:'AIRCRAFT'}
dataset={campaign:'OTTO1999',DATA_CATEGORY:'EXPERIMENT', $
         experiment:'GHOST',title:'validation', $
         type_of_data:'MEASUREMENTS'}
global=CREATE_STRUCT('pi',pi,'platform',platform,$
                     'dataset',dataset)
time={param:DINDGEN(100),$
      units:'seconds since 2000-01-01 00:00:00 UTC', $
      long_name:'time', short_name:'time',flag:'NONE'}
time=create_struct(time,valid_param(time))
dim_press={name:'press'}
status_press={dim:dim_press}
press={param:findgen(10),units:'hPa',long_name:'pressure', $
       sample_interval:1.0,$
       short_name:'press',flag:'NONE',status:status_press}
press=create_struct(press,valid_param(press))
dim_n2o={name:['press','time']}
status_n2o={dim:dim_n2o}
n2o={param:FINDGEN(10,100),units:'ppm',long_name:'N2O', $
     sample_interval:1.0,$
     short_name:'N2O',flag:'NONE',status:status_n2o}
n2o=create_struct(n2o,valid_param(n2o))
struct=CREATE_STRUCT('!global',global,'time',time,'press',press,$
                     'n2o',n2o)

struct_tree,struct

; html_file_struct,'test.html',struct, /subtab,/border,/fixed,max=10
```

Beispiel 4.3: ICG_STRUCT: Definition: zusätzliche Abhängigkeit ausser time.

Schematischer Aufbau der Struktur aus Beispiel 4.3 auf der vorherigen Seite:



Definition einer Datenstruktur ohne die Abhängigkeit von time

Im Unterschied zu Beispiel 4.3 auf Seite 38 ist in diesem Beispiel keine Abhängigkeit zur Zeit definiert. Aus diesem Grund müssen alle Abhängigkeiten in der STATUS-Struktur definiert werden.

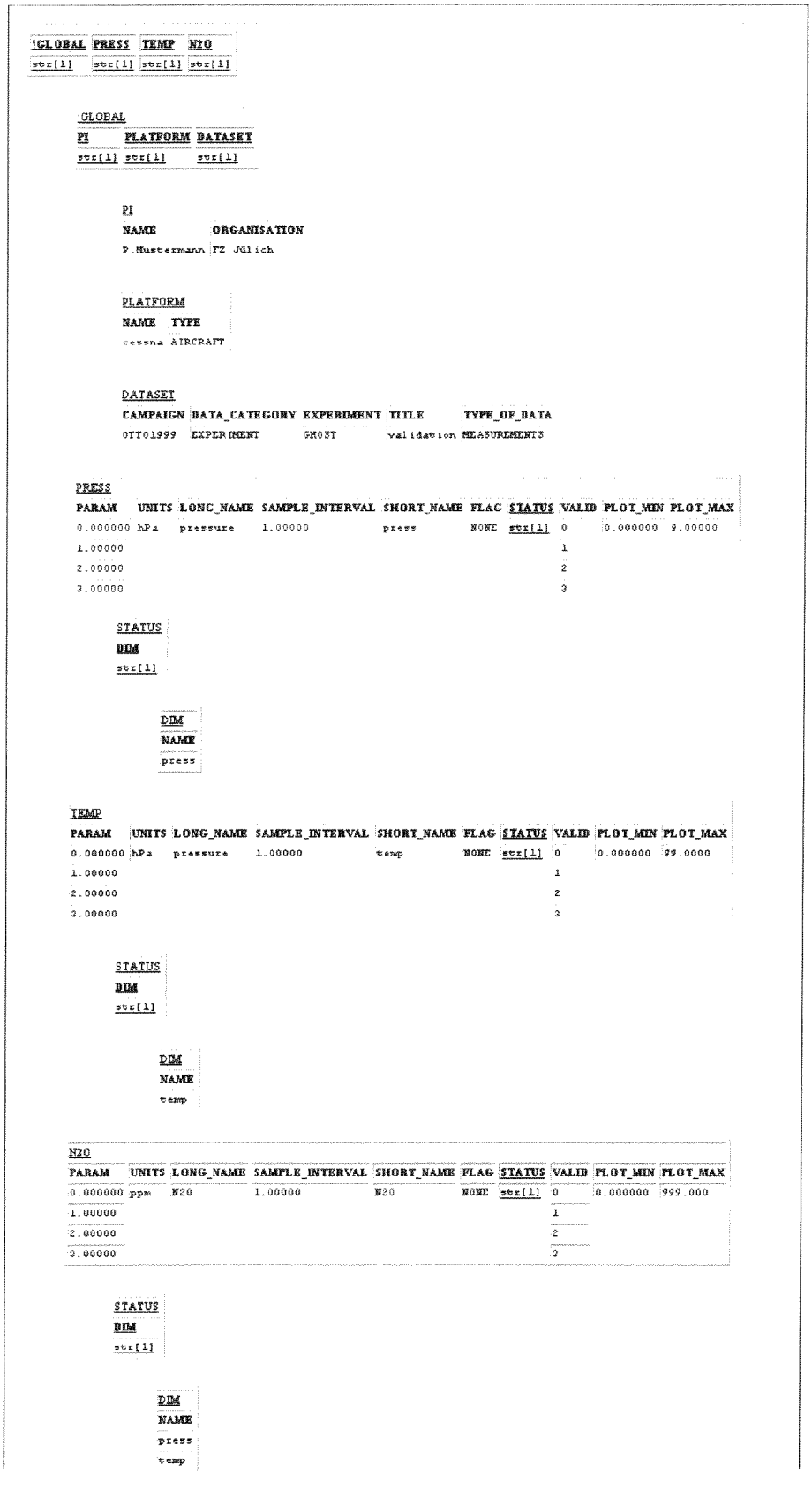
```
pi={name:'P.Mustermann',organisation:'FZ Jülich'}
platform={name:'cessna',type:'AIRCRAFT'}
dataset={campaign:'OTTO1999',DATA_CATEGORY:'EXPERIMENT', $
          experiment:'GHOST',title:'validation', $
          type_of_data:'MEASUREMENTS'}
global=CREATE_STRUCT('pi',pi,'platform',platform, $
                     'dataset',dataset)
dim_press={name:'press'}
status_press={dim:dim_press}
press={param:findgen(10),units:'hPa',long_name:'pressure', $
        sample_interval:1.0,$
        short_name:'press',flag:'NONE',status:status_press}
press=create_struct(press,valid_param(press))
dim_temp={name:'temp'}
status_temp={dim:dim_temp}
temp={param:findgen(100),units:'hPa',long_name:'pressure', $
       sample_interval:1.0,$
       short_name:'temp',flag:'NONE',status:status_temp}
temp=create_struct(temp,valid_param(temp))
dim_n2o={name:['press','temp']}
status_n2o={dim:dim_n2o}
n2o={param:FINDGEN(10,100),units:'ppm',long_name:'N2O', $
      sample_interval:1.0,$
      short_name:'N2O',flag:'NONE',status:status_n2o}
n2o=create_struct(n2o,valid_param(n2o))
struct=CREATE_STRUCT('!global',global,'press',press,'temp',temp,$
       'n2o',n2o)

struct_tree,struct

html_file_struct,'test.html',struct, /subtab,/border,/fixed,max=4 ,$
                  /block
```

Beispiel 4.4: ICG_STRUCT: Definition: ohne Abhängigkeit von time.

Schematischer Aufbau der Struktur aus Beispiel 4.4 auf der vorherigen Seite:

- STRUCT
  - !GLOBAL
    - PI
      - NAME
      - ORGANISATION
    - PLATFORM
      - NAME
      - TYPE
    - DATASET
      - CAMPAIGN
      - DATA_CATEGORY
      - EXPERIMENT
      - TITLE
      - TYPE_OF_DATA
  - PRESS
    - PARAM
    - UNITS
    - LONG_NAME
    - SAMPLE_INTERVAL
    - SHORT_NAME
    - FLAG
    - STATUS
      - DIM
        - NAME
    - VALID
    - PLOT_MIN
    - PLOT_MAX
  - TEMP
    - PARAM
    - UNITS
    - LONG_NAME
    - SAMPLE_INTERVAL
    - SHORT_NAME
    - FLAG
    - STATUS
      - DIM
        - NAME
    - VALID
    - PLOT_MIN
    - PLOT_MAX
  - N2O
    - PARAM
    - UNITS
    - LONG_NAME
    - SAMPLE_INTERVAL
    - SHORT_NAME
    - FLAG
    - STATUS
      - DIM
        - NAME
    - VALID
    - PLOT_MIN
    - PLOT_MAX

HTML Ansicht der Struktur aus Beispiel 4.4 auf Seite 40:



'GLOBAL | PRESS | TEMP | N2O
str[1] | str[1] | str[1] | str[1]

'GLOBAL

PI | PLATFORM | DATASET
str[1] | str[1] | str[1]

PI

NAME | ORGANISATION
P.Mustermann | FZ Jülich

PLATFORM

NAME | TYPE
cessna | AIRCRAFT

DATASET

CAMPAIGN | DATA_CATEGORY | EXPERIMENT | TITLE | TYPE_OF_DATA
0TT01999 | EXPERIMENT | GHOST | validation | MEASUREMENTS

PRESS

PARAM	UNITS	LONG_NAME	SAMPLE_INTERVAL	SHORT_NAME	FLAG	STATUS	VALID	PLOT_MIN	PLOT_MAX
0.000000	hPa	pressure	1.00000	press	NONE	str[1]	0	0.000000	9.00000
1.00000							1		
2.00000							2		
3.00000							9		

STATUS

DIM
str[1]

DIM

NAME
press

TEMP

PARAM	UNITS	LONG_NAME	SAMPLE_INTERVAL	SHORT_NAME	FLAG	STATUS	VALID	PLOT_MIN	PLOT_MAX
0.000000	hPa	pressure	1.00000	temp	NONE	str[1]	0	0.000000	99.0000
1.00000							1		
2.00000							2		
3.00000							3		

STATUS

DIM
str[1]

DIM

NAME
temp

N2O

PARAM	UNITS	LONG_NAME	SAMPLE_INTERVAL	SHORT_NAME	FLAG	STATUS	VALID	PLOT_MIN	PLOT_MAX
0.000000	ppm	N2O	1.00000	N2O	NONE	str[1]	0	0.000000	999.000
1.00000							1		
2.00000							2		
3.00000							3		

STATUS

DIM
str[1]

DIM

NAME
press
temp

4.2 Bearbeitung der ICG-Daten-Struktur

Da die ICG-Daten-Struktur eine Struktur mit Unterstrukturen enthält, werden Strukturoperationen auf den Unterstrukturebenen benötigt. Hierzu wurden die Funktionen `add_tag`, `delete_tag`, `rename_tag` und `replace_tagvalue` geschrieben. Diese Module sind in der Lage, auf jeder Strukturebene Strukturen oder Werte zu ändern. Weiterhin soll es mit einfachen Funktionsaufrufen möglich sein, Information aus der Struktur zu erhalten. Im Folgenden werden einige spezielle Routinen zum Bearbeiten der ICG-Daten-Struktur vorgestellt.

4.2.1 Hilfsmittel zum generellen Bearbeiten von Strukturen

- **add_tag()** Diese Funktion kann einen Tag-Namen oder eine Struktur an eine vorhandene Struktur anhängen
- **delete_tag()** Diese Funktion kann einen Tag-Namen aus einer Struktur entfernen
- **del_tag2()** Diese Funktion kann einen Tag-Namen aus einem Struktur-Array entfernen
- **is_tag()** Diese Funktion ermittelt ob ein Tag-Name in einer Struktur vorhanden ist.
  z.B.: `PRINT,is_tag(s.value,'status')`
- **tag_position()** Diese Funktion ermittelt die Position eines Tags in einer Struktur.
  z.B.: `PRINT,tag_position(s,'value')`
  Mit Hilfe dieser Zuordnungsnummer kann man direkt auf einen Tag-Namen einer Struktur zugreifen.
  z.B.: `HELP, s.(tag_position(s,'value')),/str`
- **rename_tag()** Diese Funktion kann den Tag-Namen einer Struktur umbenennen.
- **replace_tagvalue()** Diese Funktion kann den Tag-Namen einer Struktur durch einen anderen bzw. dessen Werte ersetzen.
- **text2tagname()** Diese Funktion übersetzt einen text in einen verwendbaren Tag-Namen, d.h. alle nicht alphanumerischen Zeichen außer '_' werden in '_' übersetzt.
- **is_structure()** Diese Funktion ermittelt, ob die abgefragte Variable eine Struktur ist.
- **reform_struct()** Diese Funktion verwendet `REFORM`, um die Dimensionierung einer Struktur oder deren Tag-Namen zu ändern.
- **size_struct()** Diese Funktion wendet die `SIZE()` Funktion auf alle Einträge einer Struktur an.
- **struct_tree** Mit dieser Routine kann man eine Struktur als Baum darstellen.
- **struct2htmltable() / html_file_struct** Diese Funktion/Prozedur ist für die Visualisierung einer Struktur in html praktisch. Dadurch kann man sich über Definitionen einen Überblick verschaffen.

```
hmtl=struct2htmltable(s,/subtables,/border,/blockquote)
html_file_struct,'out.html',s,/subtables,/border,/blockquote
```

4.2.2 Informationen zur ICG-Daten-Struktur

Es gibt eine Reihe von Funktionen, die Information über den inneren Aufbau einer ICG-Daten-Struktur geben.

q_icg_struct()

Diese Funktion überprüft ob in der ICG-Daten-Struktur alle benötigten Attribute zur Beschreibung eines Datensatzes definiert sind. Fehlende Eintragungen werden am Bildschirm angezeigt.

is_icg_struct()

Diese Funktion überprüft ob der Eintrag !GLOBAL in der Struktur definiert ist. Dadurch wird erkannt, ob die in Frage stehende Struktur eine ICG-Daten-Struktur ist.

get_tag_and_short_names()

Diese Funktion erstellt eine 1:1 Zuordnungstabelle zwischen den Short-Namen und den Tag-Namen.

```
ts=get_tag_and_short_names(s)
idx=where(ts.short_name eq 'N2O')
help,s.(tag_position(s,ts.tag_name[idx])),/str
```

Beispiel 4.5: ICG_STRUCT: get_tag_and_short_names()

pos_icg_struct()

Routinen sollten über den *short_name* eine Zuordnung zu den Daten erlangen, da dadurch eine eindeutige Zuordnung gegeben ist. Der Tag-Name braucht keinen Bezug zu dem *short_name* zu haben. Mit der Funktion `pos_icg_struct` wird die Position eines *short_names* in der ICG-Daten-Struktur festgestellt. Diese Funktion kann auch direkt die Anfrage von mehreren *short_names* bearbeiten.

```
pos=pos_icg_struct(icg,'Fluoreszenz')
HELP,icg.(pos),/str
```

Beispiel 4.6: ICG_STRUCT: pos_icg_struct

get_status_from_icg_struct()

Diese Funktion liefert die STATUS-Struktur (siehe Definition 4.1.4 auf Seite 32) einer ICG-Daten-Struktur zurück. Dadurch erhält man Informationen, welche Koordinatenvariablen, Dimensionen u.a. in der Struktur vorhanden sind.

get_icgs_coord_var_name()

Diese Funktion gibt die Namen der Koordinatenvariablen, die für die Parameter in der ICG-Daten-Struktur verwendet werden, zurück.

convert_icgs_name

Diese Funktion gibt u.a. den long_name und die units aus einer ICG-Daten-Struktur zurück.

4.2.3 Hilfsmittel zum Bearbeiten/Erstellen einer ICG-Daten-Struktur

Im folgenden sind einige Module beschrieben, die das Bearbeiten oder Erstellen einer Struktur vereinfachen. Die verwendeten Beispiele sind voneinander abhängig.

Übertragung der Attribute von einer ICG-Daten-Struktur auf eine andere

Für den Fall, dass gleichartig aufgebaute Datenstrukturen verwendet werden, besteht die Möglichkeit der Übernahme der Attribute aus einer vorhandenen Struktur.

Beispiel:
Das Experiment GHOST nimmt an einer Meß-Kampagne teil. Es wird immer auf dem gleichen Flugzeug eingesetzt und es misst immer die gleichen Parameter. Dann sind einige Attribute der einzelnen Mess-Datensätze immer gleich. Alle diese Konstanten kann man dann in einer Struktur zusammenfassen.

```
pi={name:'P.Mustermann',organisation:'FZ Jülich'}
platform={name:'cessna',type:'AIRCRAFT'}
dataset={campaign:'OTTO1999',DATA_CATEGORY:'EXPERIMENT',$
         experiment:'GHOST',title:'validation',$
         type_of_data:'MEASUREMENTS'}
global=CREATE_STRUCT('pi',pi,'platform',platform,'dataset',dataset)
n2o={units:'ppm',$
     sample_interval:1.0,$
     long_name:'mixing ratio',short_name:'N2O',$
     flag:'NONE'}
time={units:'seconds since 2000-01-01 00:00:00 UTC',$
      long_name:'time',short_name:'time',flag:'NONE'}
struct=CREATE_STRUCT('!global',global,'time',time,'n2o',n2o)
```

Beispiel 4.7: ICG_STRUCT: Definition der konstanten Attribute

Die sich ändernden Daten faßt man in einer anderen Struktur zusammen. In diesem Beispiel misst das Gerät *N2O*. Durch die Auswertung der Daten kann man die Standardabweichung noch mit angeben *N2O@STDEV*. Eine ICG-Daten-Struktur lässt sich nun auch folgendermaßen zusammensetzen.

```
n=100
data=create_struct('time',dindgen(n),'N2O',sin(findgen(n)),$
     'N2O@STDEV',findgen(n)/10.)
icg=struct2icg_struct(data,struct,/set_flag)
```

Beispiel 4.8: ICG_STRUCT: Übertragen von Attributen

Die Verknüpfung der Strukturen mit `struct2icg_struct` erfolgt über die Tag-Namen der Strukturen. Das Keyword */set_flag* bewirkt, dass das Attribut flag von *N2O* von *'NONE'* auf *'STDEV'* umgesetzt wird.

Eine weitere Möglichkeit für das einfache Erstellen einer ICG-Daten-Struktur ist durch die Verwendung der Funktion `inherit_icg_struct` gegeben. Mit Hilfe dieser Funktion kann man die Attribute eines Parameters der ICG-Daten-Struktur auf einen anderen Parameter übertragen. Im vorliegendem Fall habe ich schon einen Parameter *N2O* in der Datenstruktur. Der neue Parameter soll die gleichen Attribute erhalten.

```
n=100
inh=inherit_icg_struct(icg,['time','N2O'],to_names=['time','N2'])
data=create_struct('time',dindgen(n),'N2',sin(findgen(n)),$
     'N2@STDEV',findgen(n)/10.)
icg=struct2icg_struct(data,inh,/set_flag)
```

Beispiel 4.9: ICG_STRUCT: Übertragen von Attributen mit inherit_icg_struct

Hinzufügen weiterer Parameter-Strukturen

Um eine zusätzliche Parameter-Struktur (im Beispiel 4.10 die Messgrösse F12 bzw. im Beispiel 4.11 die Messgrösse F11) in der ICG-Daten-Struktur zu erzeugen, gibt es mehrere Möglichkeiten.

```
icg=CREATE_STRUCT(icg,'F12',icg.n2)
icg.f12.short_name='F12'
icg.f12.units='ppt'
```

Beispiel 4.10: ICG_STRUCT: Zusätzliche Parameter Struktur

Oder:

```
F11={param: findgen(n) ,$
      units:'ppt',$
      sample_interval:1.0,$
      long_name:'mixing ratio',short_name:'F11',$
      flag:'NONE'}
icg=CREATE_STRUCT(icg,'F11',F11)
```

Beispiel 4.11: ICG_STRUCT: Zusätzliche Parameter Struktur als Unter Struktur

Umwandeln einer ICG-Daten-Struktur in eine Struktur ohne Unterstrukturen (Ein-Ebenen-Struktur) und umgekehrt

In einigen Fällen, z.B.: bei der Verwendung des Tabellen widgets wird eine Struktur ohne Unterstrukturen benötigt. Daher wurden einige Mechanismen entwickelt, die eine ICG-Daten-Struktur in eine Ein-Ebenen-Struktur übersetzen bzw. diese Ein-Ebenen-Struktur in eine ICG-Daten-Struktur übersetzen.

Es gibt zwei Möglichkeiten: Kommt es darauf an, dass man die Werte aller Parameter indizieren möchte, verwendet man `icg_struct2v_struct`. Damit erzeugt man eine array-orientierte Struktur. Mit `icg_strut2struct` erfolgt die Array-Orientierung auf den Tag-Namen, wodurch auch unterschiedlich dimensionierte Parameter in einer Ein-Ebenen-Struktur dargestellt werden können. Das Beispiel zeigt in diesem Fall die Anwendung von `icg_struct2struct` bzw. die Umkehrung mit `struct2icg_struct`. Da in dieser Weise flags (siehe Kapitel 4.1.3 auf Seite 31) einfach dazugefügt werden können, bedarf es der Funktion `set_icg_struct_flags` (siehe Kapitel 4.2.3 auf Seite 49), die dann die Flag-Namen nachträglich in die Struktur einträgt.

Aus jeder ICG-Daten-Struktur kann man auch eine Ein-Ebenen-Struktur erzeugen.

```
data=icg_struct2struct(icg)
struct_tree,data
```

Beispiel 4.12: `ICG_STRUCT: icg_struct2struct`

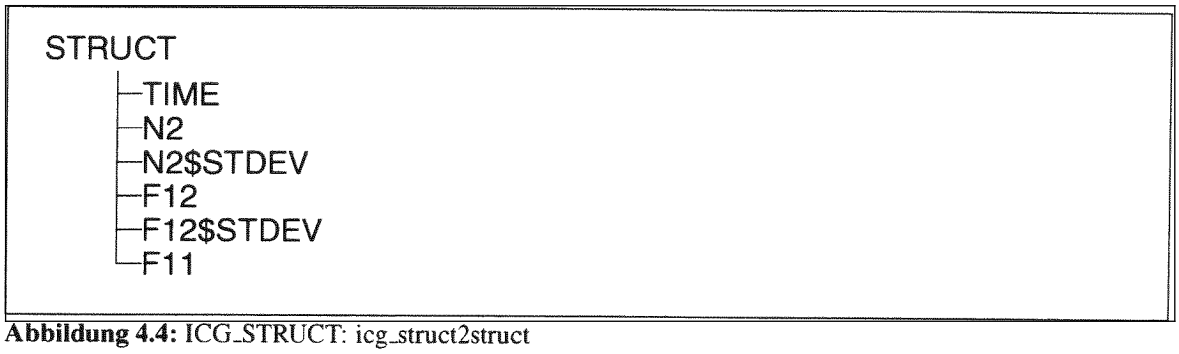


Abbildung 4.4: ICG_STRUCT: icg_struct2struct

Dieser Vorgang lässt sich, siehe nachfolgendes Beispiel auch wieder umkehren.

```
data=icg_struct2struct(icg)
icg2=struct2icg_struct(data,icg)
struct_tree,icg2
```

Beispiel 4.13: ICG_STRUCT: struct2icg_struct

```
STRUCT
    ├─IGLOBAL
    │  ├─PI
    │  │  ├─NAME
    │  │  └─ORGANISATION
    │  ├─PLATFORM
    │  │  ├─NAME
    │  │  └─TYPE
    │  └─DATASET
    │     ├─CAMPAIGN
    │     ├─DATA_CATEGORY
    │     ├─EXPERIMENT
    │     ├─TITLE
    │     └─TYPE_OF_DATA
    ├─TIME
    │  ├─SHORT_NAME
    │  ├─UNITS
    │  ├─LONG_NAME
    │  ├─FLAG
    │  ├─PARAM
    │  ├─VALID
    │  ├─PLOT_MIN
    │  └─PLOT_MAX
    ├─N2
    │  ├─SHORT_NAME
    │  ├─UNITS
    │  ├─SAMPLE_INTERVAL
    │  ├─LONG_NAME
    │  ├─FLAG
    │  ├─PARAM
    │  ├─VALID
    │  ├─PLOT_MIN
    │  ├─PLOT_MAX
    │  └─STDEV
    ├─F12
    │  ├─SHORT_NAME
    │  ├─UNITS
    │  ├─SAMPLE_INTERVAL
    │  ├─LONG_NAME
    │  ├─FLAG
    │  ├─PARAM
    │  ├─VALID
    │  ├─PLOT_MIN
    │  ├─PLOT_MAX
    │  └─STDEV
    └─F11
       ├─UNITS
       ├─SAMPLE_INTERVAL
       ├─LONG_NAME
       ├─SHORT_NAME
       ├─FLAG
       ├─PARAM
       ├─VALID
       ├─PLOT_MIN
       └─PLOT_MAX
```

Abbildung 4.5: ICG_STRUCT: struct2icg_struct

set_icg_struct_flags()

Diese Funktion setzt das flag-Attribut jedes Parameters auf die in der ICG-Daten-Struktur unter jedem Parameter vorhandenen flags um.

4.2.4 Anwendung eines Index-Felds auf eine ICG-Daten-Struktur

In einer ICG-Daten-Struktur kann man alle Parameter eines Messtages abbilden. Möchte man bestimmte Abhängigkeiten der Parameter darstellen, kann man ein Index-Feld der Abhängigkeit auf die ICG-Daten-Struktur anwenden.

icg_struct_idx()

Häufig kommt es vor, dass man die Anzahl der Daten in Abhängikeit einer Größe veranschaulichen möchte. Mit der `WHERE` oder `SORT` Funktion kann man ein Index-Feld erzeugen. Solch ein Index-Feld kann man auf die ganze Struktur anwenden, sofern sie nur eine Koordinaten-Variable hat. Bei Zeitserien ist das in der Regel so.

```
pos=pos_icg_struct(icg,'Fluoreszenz')
idx=where(icg.(pos).param lt 10000.)
data=icg_struct_idx(icg,idx)
```

Beispiel 4.14: ICG_STRUCT: icg_struct_idx

icgs_filter()

Diese Funktion gibt ein Index-Feld zurück, das von einer in Text-Form angegebenen Bedingung abhängig ist.

```
icg=read_ncdf('kurs_6_1.nc')
idx=icgs_filter(icg,'Fluoreszenz LT 10000.')
st1=icg_struct_idx(icg,idx)
```

Beispiel 4.15: ICG_STRUCT: icgs_filter

icg_struct4v_struct() (nur für Vektordaten)

Diese Funktion erlaubt es, eine ICG-Daten-Struktur in Form einer Tabelle mit Hilfe eines widgets anzusehen und zu bearbeiten. Das Resultat wird als ICG-Daten-Struktur zurückgegeben.

```
icg2=icg_struct4v_struct(icg)
```

Beispiel 4.16: ICG_STRUCT: icg_struct4v_struct()

4.2.5 Erstellung der valid-Vektoren auf den Parameter-Ebenen

valid_param()

Diese Funktion gibt eine Teil-Struktur zurück, mit der man die gültigen bzw. nicht gültigen Parameterwerte identifizieren kann. Sie muß auf eine einzelne Parameter-Struktur angewendet werden.

```
value={param:FINDGEN(10),units:'ppm',long_name:'N2O', $
       short_name:'N2O',flag:'NONE',missing_value:-999.9}
value=create_struct(value,valid_param(value))
```

Beispiel 4.17: ICG_STRUCT: valid_param()

icg_struct_param_valid()

Mit dieser Funktion wird die valid-Struktur eines Parameters (*short_name*) aus der ICG-Daten-Struktur neu bestimmt.

update_valid_index()

Mit dieser Funktion wird die valid-Struktur eines oder mehrerer Parameter in der ICG-Daten-Struktur aktualisiert.

4.3 Anwendung der ICG-Daten-Struktur

Dieses Kapitel beschäftigt sich mit der Verwendung der ICG-Daten-Struktur bei Datei-Operationen und dem Prüfen und Abbilden der darin enthaltenen Daten. Die Zitate der zugrunde liegenden statistischen Funktionen sind in den jeweiligen Quelltexten der Routinen aufgeführt.

4.3.1 Datei-Operationen

read_ncdf(), write_ncdf

Mit diesen beiden Routinen `read_ncdf` und `write_ncdf` ist das Lesen bzw. Speichern der ICG-Daten-Struktur einer netCDF-Datei möglich. Da netCDF Dateien sehr groß werden können, erlaubt `read_ncdf` beim Lesen die Einschränkung auf einzelne Parameter der Datensätze. Dabei werden jeweils die dazugehörigen Koordinatenvariablen (Bezugs-Dimensions-Variable, z.B. time) automatisch mitausgelesen.

Da das Lesen ganzer Dateien mitunter lange dauert, steht ein Cache-Mechanismus zur Verfügung, der global durch die Environment-Variablen *local_cache_path*, *no_cache* gesteuert wird. Wird *no_cache* mit 1 gesetzt, ist das 'cachen' abgestellt. Mit der Env-Variable *local_cache_path* kann man den Pfad für das 'Cachen' vorgeben. Standardpfad ist das Verzeichnis *idl_cache* unter *idl_source*. Alternativ stehen Steuerworte zur Verfügung.

```
icg=read_ncdf('kurs_6_1.nc')
icg=read_ncdf('kurs_6_1.nc',/new_cache)
icg=read_ncdf('kurs_6_1.nc',/no_cache)
icg=read_ncdf('kurs_6_1.nc',['Fluoreszenz','Spiegelposition'])
```

Beispiel 4.18: ICG_STRUCT: read_ncdf()

Mit der Schreibroutine `write_ncdf` kann man entweder Daten in eine neue Datei schreiben oder in einer vorhandenen Datei Daten überschreiben. Zum Überschreiben muß man das Steuerwort */overwrite* setzen. Falls man eine nicht der Norm entsprechende ICG-Daten-Struktur hat, kann man mit dem Steuerwort */ignore_test* das Testen mit der Funktion `q_icg_struct` verhindern.

```
write_ncdf,'test.nc',icg
write_ncdf,'test.nc',icg,/overwrite
write_ncdf,'test.nc',icg,/ignore_test
```

Beispiel 4.19: ICG_STRUCT: write_ncdf

read_enz(), write_enz

Routine zum Einlesen bzw. Schreiben des ASCII Austausch Datenformats vom ICG-3 dem ENZ-Format, (siehe [*H. London et al.*(1992)]).

```
icgs=read_enz('test.enz')
write_enz,'test2.enz',icgs
```

Beispiel 4.20: ICG_STRUCT: read_enz(), write_enz

ames2icgs()

Routine zum Einlesen der NASA Ames Datenfiles [*S. E. Gaines et al.*(1990)] mit Hilfe der Steve Gaines IDL Bibliothek in die ICG-Daten-Struktur. Die NASA Ames Datenfiles sind ASCII Dateien die häufig bei internationalen Messkampagnen zum Austauschen von Daten verwendet werden.

4.3.2 Prüfen, Abbilden

x_syncronize, icgs_timeshift()

Häufig ist es ein Problem festzustellen, ob abgegebene Daten verschiedener Quellen wirklich auf der gleichen Zeitbasis beruhen. Dieses Widget hilft, evtl. Zeitunterschiede aufzuspüren.

Mit der graphischen interaktiven Benutzer-Schnittstelle (widget) `x_synchronize` kann man zwei Datenreihen zeitlich gegeneinander verschieben und sich den Verlauf des Korrelationskoeffizienten gegen die Verschiebung ansehen. Das kann dazu dienen, eventuelle Inkonsistenzen in den Zeitachsen der Daten herauszufinden. Die Funktion `icgs_timeshift` ermittelt diesen Sachverhalt ohne interaktive Benutzer-Schnittstelle.

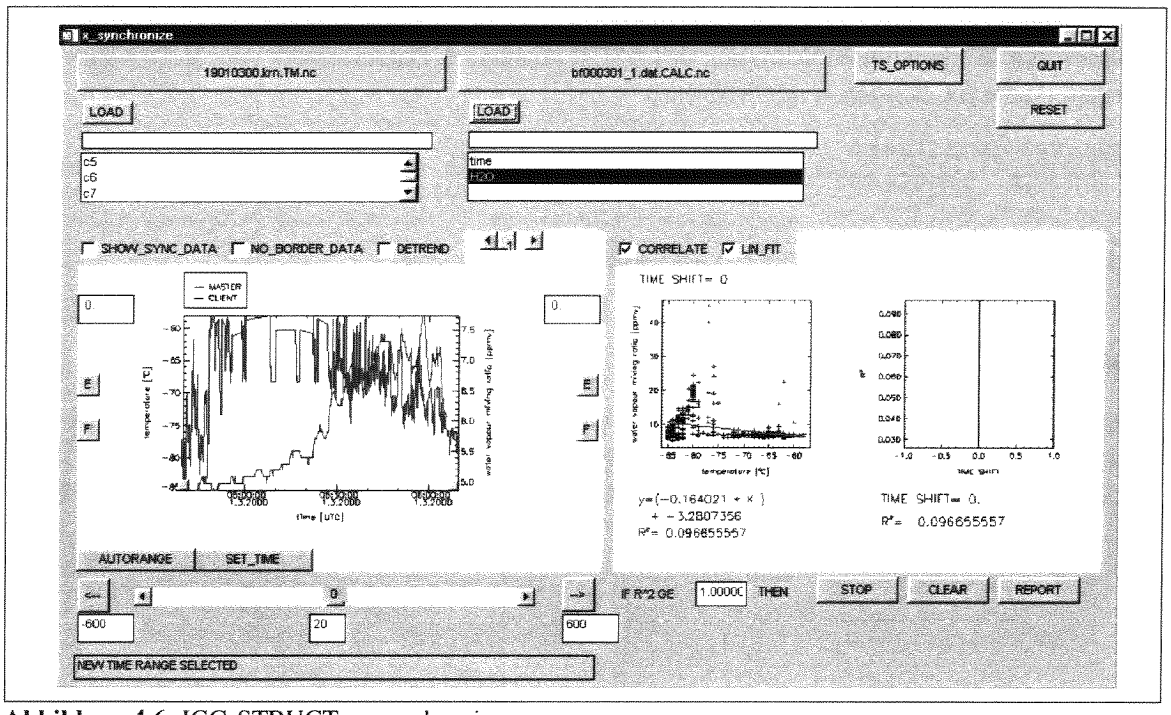


Abbildung 4.6: ICG_STRUCT: x_synchronize

```
x_synchronize,master_icg_struct,client_icg_struct
```

```
result = icgs_timeshift(master, client, ['HCHO (DOAS)', $
 'HCHO (Hantzsch)'], /VALID)
```

icg_ts_sync()

Mit der Funktion icg_ts_sync wird die Zeit-Synchronisation der Daten einer ICG-Daten-Struktur ermöglicht. Das Ergebnis ist eine neue ICG-Daten-Struktur.
Mit `xp_icg_ts_sync` steht ein grafisches Benutzerinterface für diese Funktion zur Verfügung.

```
new=icg_ts_sync(icg1,icg2,/linear_with_limit,interpolation_limit=10)
```

Beispiel 4.21: ICG_STRUCT: icg_ts_sync

icgs_correlate()

Diese Funktion bestimmt den linearen Korrelationskoeffizienten.

icgs_mean_dc()

Diese Funktion bestimmt den mittleren Tagesgang von einem Parameter der ICG-Daten-Struktur.

icgs_slice()

Diese Funktion gibt eine Untermenge der Parameter einer ICG-Daten-Struktur in Abhängigkeit eines Index zurück.

```
a = read_ncdf('berlioz_o3.nc')
b = icgs_slice(a, icgs_filter(a, 'O3 GT 10'))
```

Beispiel 4.22: ICG_STRUCT: icgs_slice

icgs_stat()

Diese Funktion ergibt die Anwendung einiger statistischer Funktionen auf die Parameter der ICG-Daten-Struktur. U.a. wird der Mittelwert, die Standardabweichung und der Median gebildet.

icgs_timeshift()

Diese Funktion ermittelt den Zeitversatz mittels linearer Korrelation zwischen zwei Zeitserien.

icgs_time_filter()

Diese Funktion gibt einen Index, der auf verschiedenen Zeit-Kriterien beruht zurück.

```
o3_icgs = read_ncdf('berlioz_o3.nc')
time_ind = icgs_time_filter(o3_icgs, $
  TIME = ['10:00:00', '12:00:00'])
```

Beispiel 4.23: ICG_STRUCT: icgs_time_filter

icgs_time_slice()

Diese Funktion gibt eine Untermenge der Parameter einer ICG-Daten-Struktur basierend auf einem definierten Zeitintervall zurück

icgs_concat()

Diese Funktion verbindet zwei ICG-Daten-Strukturen zu einer ICG-Daten-Struktur.

ncdf_files2icg_struct()

Mit dieser Funktion können netCDF Dateien mit gleichen Koordinatenvariablen in eine ICG-Daten-Struktur abgebildet werden (siehe Beispiel 6.7 auf Seite 134. Durch das Keyword `/concat_names` werden Parameter gleichen Namens auf einem Parameter abgebildet (siehe Beispiel 6.8 auf Seite 135).

x_icgs_add_quality

Dieses Programm ist eine graphische Benutzerschnittstelle um den QUALITY-Flag von 0 (gut) oder 32 (schlecht) in einer ICG-Daten-Struktur zu setzen.

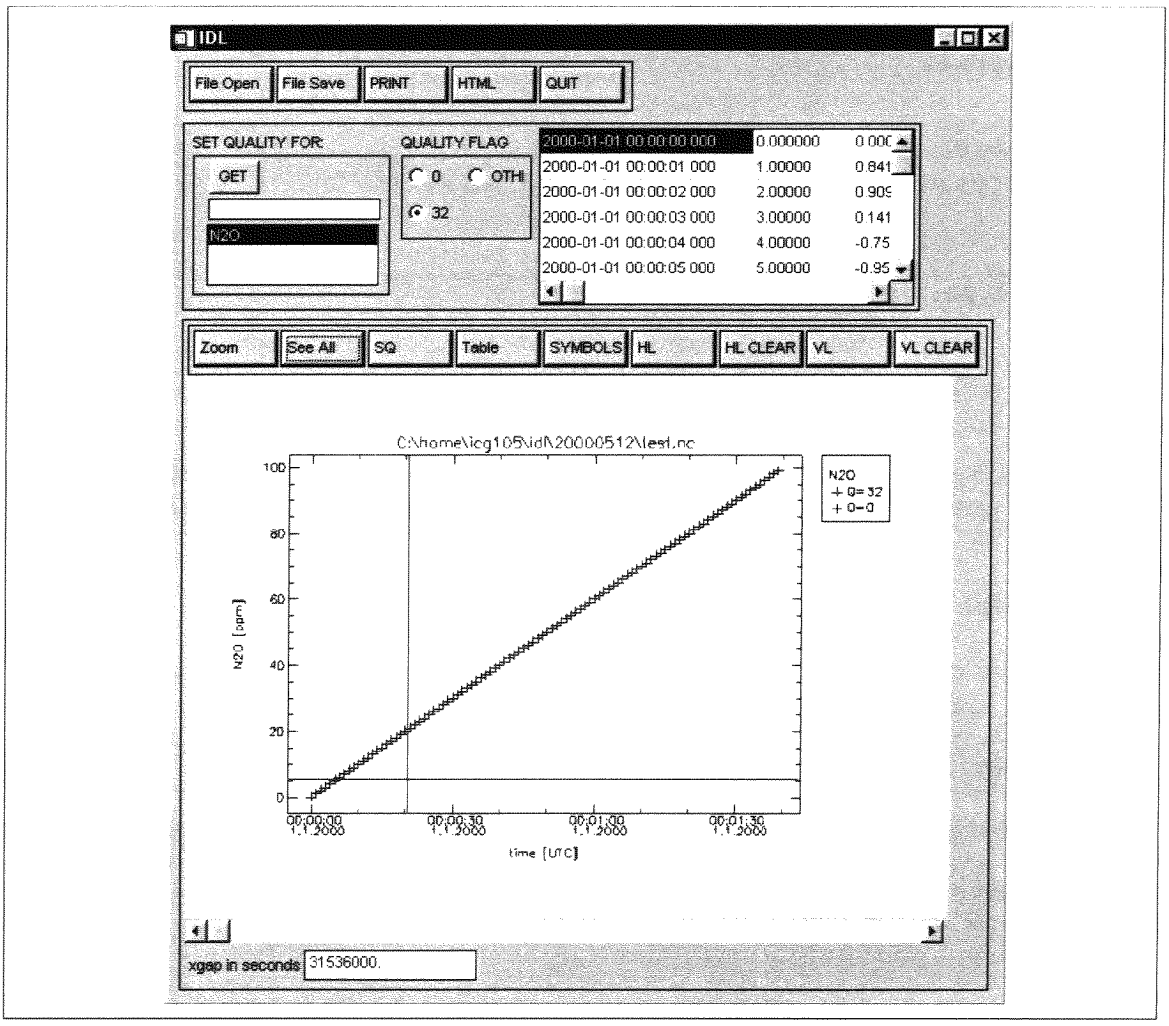


Abbildung 4.7: ICG_STRUCT: x_icgs_add_quality (Setzen von QUALITY)

Kapitel 5

Die PLOT-Umgebung

Aufgrund der Schwierigkeiten, die jeder Anwender bei der Erstellung einer gut aussehenden Grafik hat, wurde eine PLOT-Umgebung entwickelt, die durch geschickte Voreinstellungen qualitativ hochwertige Grafiken erstellt. Diese Voreinstellungen werden in Form einer Struktur, im weiteren PLOT-Struktur genannt, an alle PLOT-Module weitergereicht; der Anwender muss lediglich wenige Schlüsselworte definieren, um einen guten Ausdruck zu erhalten.

Mittlerweile gibt es Module für die Darstellung von 2D, 3D und 4D-Daten mit und ohne Kartenprojektionen.

Für die PLOT-Struktur gibt es eine Reihe von Widgets, die die Erstellung einer Grafik noch weiter vereinfachen. Diese Widgets sind in der Lage, die PLOT-Struktur zu verarbeiten und die Änderungen an dieser Struktur als Information in das Script abzulegen. Danach wird der Aufruf des Widgets aus dem Script entfernt; die Erstellung des Bildes ist dann komplett im Script enthalten und reproduzierbar.

Diese Dokumentation ist so gegliedert, dass zunächst alle Optionen der PLOT-Umgebung für alle Module beschrieben werden. Danach werden alle Module und deren Möglichkeiten beschrieben.

Am Ende werden einige Anwendungen aus der Praxis gezeigt.

5.1 Allgemein

Syntax:

Ein PLOT-Script ist grundsätzlich aus zwei Teilen aufgebaut. Im ersten Teil werden alle Punkte definiert, die sich mit der Steuerung des Ausgabegerätes befassen (Drucker oder Bildschirm). Im Zweiten wird das Aussehen des Plots festgelegt.
Das einfachste denkbare Script besteht aus folgenden vier Zeilen:

```
plotprepare,plot  ; @init
plotinit,plot
plotxy,plot,x=x,y=y
plotend,plot
```

Beispiel 5.1: PLOT Umgebung: prinzipieller Aufbau

Dabei umrahmen die Befehle `plotprepare (@init)` und `plotinit` den oben erwähnten ersten Teil, während zwischen `plotinit` und `plotend` der eigentliche Plot definiert wird. Man muss deswegen darauf achten, dass alle Befehle, die sich mit der Steuerung des Ausdrucks befassen (z.B. *plot.psflag* oder *plot.landscape*), vor `plotinit` stehen, da sie sonst keine Wirkung

haben. Andererseits kann man natürlich nicht mit `plotxy` einen plot ausführen, bevor das Ausgabegerät definiert ist. Der Befehl `plotxy` darf also nur nach `plotinit` stehen. `plotxy` steht hier stellvertretend für alle anderen PLOT-Module. Derzeit gibt es die Module: `plot_probability`, `plotxy`, `plot2d`, `plotxy_2d`, `plotcell`, `plotmap`, `plot3d`. Diese Module sind aufeinander abgestimmt. Dadurch kann man durch Austausch eines Moduls leicht eine andere Grafik erstellen.

plotprepare,plot

Die gesamte Ausführung einer Grafik wird durch Steuervariablen festgelegt. Diese Steuervariablen sind in der PLOT-Struktur zusammengefasst. Durch `plotprepare,plot` wird die Struktur *plot* angelegt, und die Steuervariablen werden mit ihren Voreinstellungen belegt. Mit Ausnahme bei der Verwendung der `xp_widgets` ist der Variablen Name der Struktur hier *plot* frei wählbar. Die Voreinstellungen sind so definiert, dass eine standardisierte Grafik entsteht, ohne dass der Benutzer irgendetwas (z.B. Schriftgrösse, Strichdicke oder Symbolnummer) eingeben muss. Wenn er trotzdem Details ändern will, kann er dies durch Änderung in der PLOT-Struktur tun, siehe Kapitel 5.2 auf Seite 58.

Bei der Erstellung der PLOT-Struktur werden auch eine Farbtabelle und eine Vielzahl von Symbolen erzeugt. Die untersten 20 Farb-Indizes sind dabei für die gängigsten Farben reserviert, die der Benutzer unabhängig vom gewünschten Farbschema immer zur Verfügung hat. Der Zugriff auf diese Grundfarben ist durch die Unterstruktur *plot.color_nc* möglich (z.B. *plot.color* = *plot.color_nc.red* ergibt immer die Farbe Rot, solange man sich in der Plot-Umgebung befindet). Die Symbole erreicht man in der Unter-Struktur *plot.symbols* (z.B. *plot.symbols* = *plot.symbols.circle* stellt ein Kreis-Symbol dar).

Bisher definierte Farben und Symbole sind:

Farbe:	Name:
	schwarz
	rot
	blau
	grün
	rosa
	lila
	orange
	dunkelrot
	zyan
	gelb
	blau grün
	weiß
	hellgrau
	mittelgrau
	gold
	grün blau
	orchid
	beige

Tabelle 5.1: PLOT Umgebung: Farbtabelle, ct_ncdf

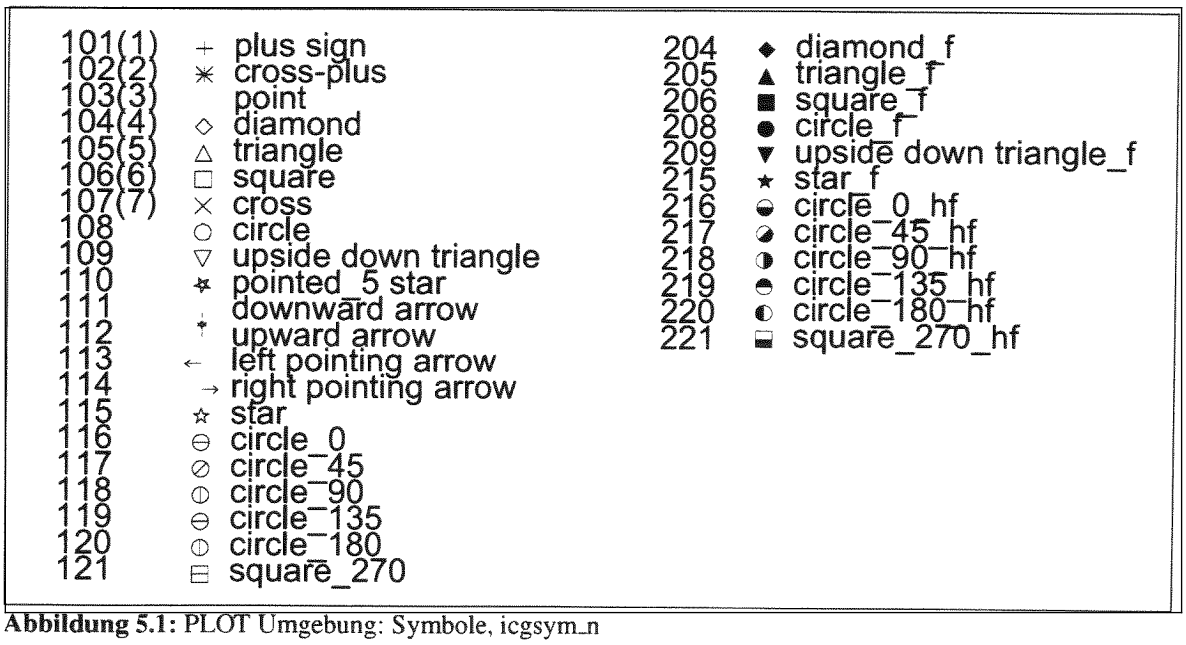


Abbildung 5.1: PLOT Umgebung: Symbole, icgsym_n

plotinit,plot

Mit dieser Zeile werden die Definitionen für die Ausgabe der Bilder für das Ausgabemedium abgeschlossen und das Ausgabemedium physikalisch initialisiert (z.B.: wird bei Postscript-Ausgabe die Postscript-Datei geöffnet).

plotxy,plot,x=x,y=y [,file=file] [,icg_struct=icg_struct] [,sx=sx,sy=sy] [,/time]

Das ist der Befehl, der die Grafik zeichnet. Die Struktur *plot* muss immer übergeben werden, `x=x` und `y=y` sind entweder der Name eines oder mehrerer Parameter aus einer netCDF-Datei, einer ICG-Daten-Struktur oder ein ein- bis mehrdimensionales Feld von Werten. `file=file` bzw. `icg_struct=icg_struct` wird benötigt, wenn man sich für die Namensübergabe entschieden hat.

`sx=sx` und oder `sy=sy` können, müssen aber nicht übergeben werden. Ist eines oder beide der Argumente vorhanden, werden Fehlerbalken in die betreffende Richtung gezeichnet.

Mit dem keyword `/time` definiert man die Bedeutung der X-Werte als Zeitwerte in Sekunden bezogen auf den 01.01.2000 00:00:00 (*julian seconds*) . Dadurch werden datumsbezogene Achsenbeschriftungen erstellt.

Die einzelnen *plot*-Steuerbefehle werden später anhand von Beispielen erklärt (siehe Beispiele 5.5 auf Seite 84)

plotend,plot

Mit diesem Befehl wird die Grafik beendet und das Device geschlossen, (z.B. die Postscript-Datei).

5.2 Aufbau der PLOT Struktur

Die PLOT-Struktur besteht aus mehreren Objekten. Diese Objekte sind in Steuermöglichkeiten für das Seitenlayout, den Titelblock, die Symbole, die Farben, die Achsen, die Zeitachse, die Legende, den Farb-Balken, die Anordnung der Bilder, das FZJ-Logo, der Fehlerbalken, der Kartenprojektionen, den speziellen Kommandos für 2D und 3D Darstellungen gegliedert. Für alle Parameter ist eine sinnvolle Voreinstellung getroffen worden. In ihrer Gesamtheit bestimmen die Einträge in der PLOT-Struktur dabei komplett das Aussehen und die Anordnung der folgenden Grafiken.
Einige der Einträge sind vordimensioniert, z.B. *plot.ytitle* ist ein Vektor mit 25 Elementen oder *plot.psym* enthält 25 x 25 Elemente. Dadurch kann man mit dem `plotxy-Modul` durch Übergabe von Datenfeldern matrixartig angeordnete Grafiken erstellen, siehe Beispiel 5.5.7 auf Seite 96. Die Indizierung der vorbelegten Felder erfolgt mit der Nummer des Bildes.
z.B.:

```
plot.ytitle[0:1,0]=['SIN','COS']
plot.psym=[0,1]
```

Der entsprechende ytitle oder das entsprechende Symbol werden in der jeweiligen Grafik verwendet. Dieser Mechanismus ist jedoch auf einige Definitionen beschränkt. Matrixartig angeordnete Grafiken erstellt man am Besten durch die Verwendung von *plot.x_interstice* bzw. *plot.y_interstice*, siehe Beispiel 5.9 auf Seite 94. In dieser Weise können alle PLOT-Module für die Ausgabe matrixartig angeordneter Bilder verwendet werden.

5.2.1 Kommandos für das Layout

Diese Kommandos werden zur Initialisierung durch `plotinit` verwendet.

Kommando:	Voreinstellung:	Bedeutung:
plot.paper_height	21.0	Papierhöhe im Querformat Einheit: cm
plot.paper_width	29.7	Papierbreite im Querformat Einheit: cm
plot.plot_height	10.85	Plothöhe im Querformat Einheit: cm
plot.plot_width	15.6	Plotbreite im Querformat Einheit: cm
plot.bottom_margin	5.0	Abstand X-Achse zum unterem Rand Einheit: cm
plot.left_margin	6.0	Abstand Y-Achse zum linken Rand Einheit: cm
plot.landscape	1	Seiten-Format: 1 Querformat, 0 Hochformat, 2 Quadratische Ausgabe im Hochformat 3 Quadratische Ausgabe im Querformat
plot.psflag	0	Ausgabe-Medium: 0 Bildschirm, 1 Postscript, 2 EPS , [10,11,12] in den Z-Buffer: 10 Bild bleibt im Z-Buffer; plotend schliesst das Device 11 letzes Bild aus dem Z-Buffer wird dargestellt: plotend schliesst das Device, 12 alles wird in den Z-Buffer geschrieben; plotend schliesst das Device nicht
plot.psfile	'idl'	Name der Ausgabedatei bei Postscript-Ausgabe
plot.stamp	1	Kommentarzeilen im Ausdruck: 0 keine, 1 Ausgabe eines Zeit-Stempels am Papierrand 2 zusätzliche Ausgabe der im Plot verwendeten Datendateinamen
plot.window	1	0 bis 31 sind normale Fenster, ab 32 ist das Ausgabefenster ein DRAW_WIDGET; das DRAW_WIDGET ist vorher zu definieren
plot.charsize	1.0	Schriftgröße der Beschriftungen
plot.page_title	''	Titel am oberen Rand auf jeder Seite
plot.page_title_alignment	0.5	Ausrichtung des Textes 0 links, 0.5 zentriert, 1 rechts
plot.page_title_color	schwarz	Farbe des Seitentitels
plot.page_title_charsize	2.	Schriftgröße des Seitentitels

Tabelle 5.2: PLOT Umgebung: Kommandos für das Layout

5.2.2 Kommandos für die Anordnung der Bilder

Diese Kommandos werden von `plotprobability`, `plotxy`, `plotxy_2d`, `plot2d`, `plotcell`, `plot3d` und `plotmap` verwendet.

Kommando:	Voreinstellung:	Bedeutung:
plot.rows	1	Horizontale Anzahl der Bilder auf einer Seite
plot.columns	1	Vertikale Anzahl der Bilder auf einer Seite
plot.new	1	Typ des nächsten Bildes: 0 Overlay, 1 neues Bild, 2 Overlay; Legendenausgabe wird komplett unterdrückt, 3 Overlay; nächste Datenreihe wird nicht in die Legende aufgenommen
plot.x_interstice	100.	prozentualer Abstand des Zwischenraums zwischen zwei übereinander angeordneten Bildern
plot.y_interstice	100.	prozentualer Abstand des Zwischenraums zwischen zwei nebeneinander angeordneten Bildern
plot.plot_frameposition [4]	[-999.]	Voreinstellung bedeutet normaler Plotbereich. Ansonsten Framekoordinaten für den nächsten plot in Verbindung mit set_frame

Tabelle 5.3: PLOT Umgebung: Kommandos für die Anordnung der Bilder

5.2.3 Kommandos für das Logo

Diese Kommandos werden von `plotprobability`, `plotxy`, `plotxy_2d`, `plot2d`, `plotcell`, `plot3d` und `plotmap` verwendet.

Kommando:	Voreinstellung:	Bedeutung:
plot.logo.title	'Forschungszentrum Jülich'	Text, der in der Logozeile dargestellt wird
plot.logo.type	0	Logo des FZJ einblenden: 0 Aus 1 'J' in der rechten oberen Ecke 2 der ganze Logo Schriftzug

Tabelle 5.4: PLOT Umgebung: Kommandos für das Logo

5.2.4 Kommandos für mehrzeiligen Titel über dem Rahmen des Plots

Diese Kommandos werden durch `plot_text` verwendet.

Kommando:	Voreinstellung:	Bedeutung:
plot.plot_title.name	"	Text für den Titel oberhalb des Plots
plot.plot_title.alignment	0.5	Ausrichtung des Titels: 0 links, linksbündig, 0.5 zentriert, 1 rechts, rechtsbündig
plot.plot_title.charsize	1.0	die Schriftgrösse des Titels
plot.plot_title.charthick	1.0	die Dicke der Schrift für den Titel
plot.plot_title.color	schwarz	die Farbe der Schrift für den Titel

Tabelle 5.5: PLOT Umgebung: Kommandos für mehrzeiligen Titel über dem Rahmen des Plots

5.2.5 Kommandos für die Grund Farben

Diese Kommandos werden von `plotprobability` und `plotxy` verwendet.

Kommando:	Wert:	Bedeutung:	R:	G:	B:	Farbe:
plot.color_nc.black	0	Farbe: schwarz	0	0	0	
plot.color_nc.red	1	Farbe: rot	255	0	0	
plot.color_nc.blue	2	Farbe: blau	0	0	255	
plot.color_nc.green	3	Farbe: grün	0	255	0	
plot.color_nc.pink	4	Farbe: pink	255	0	255	
plot.color_nc.violett	5	Farbe: violett	128	0	255	
plot.color_nc.orange	6	Farbe: orange	255	128	0	
plot.color_nc.dark_red	7	Farbe: dunkel rot	180	0	0	
plot.color_nc.cyan	8	Farbe: cyan	0	255	255	
plot.color_nc.yellow	9	Farbe: schwarz	255	255	0	
plot.color_nc.blue_green	10	Farbe: blau-grün	0	255	190	
plot.color_nc.white	11	Farbe: weiss	255	255	255	
plot.color_nc.light_grey	12	Farbe: hell-grau	200	200	200	
plot.color_nc.medium_grey	13	Farbe: mittel-grau	136	136	136	
plot.color_nc.gold	14	Farbe: gold	255	187	0	
plot.color_nc.aquamarine	15	Farbe: aquamarin	112	219	147	
plot.color_nc.orchid	16	Farbe: orchid	219	112	219	
plot.color_nc.beige	17	Farbe: beige	255	171	127	

Tabelle 5.6: PLOT Umgebung: Kommandos für die Grund Farben

5.2.6 Kommandos für die Symbole

Diese Kommandos werden von `plotprobability`, `plotxy` und `plotxy_2d` verwendet.

Kommando:	Voreinstellung:	Bedeutung:
plot.psym [25,25]	1	Die Symbol-Nummer, siehe Tabelle 5.2.6 auf der nächsten Seite
plot.color [25,25]	schwarz	Der Index der Farbpalette, siehe Tabelle 5.2.5
plot.symsize [25,25]	1.	Die Größe des Symbols
plot.linestyle [25,25]	0	Die Art der Linienzeichnung, 1 bis 4
plot.thick [25,25]	1.	Die Dicke der Linie

Tabelle 5.7: PLOT Umgebung: Kommandos für die Symbole

Zu den von IDL vorgegeben Symbolen sind zusätzliche Symbole definiert worden. Die Werte in runden Klammern () sind Symbol-Kennziffern, die das IDL zur Verfügung stellt. Alle anderen Symbole sind in der Routine `icgsym_n` definiert worden. Werte von 100 bis 200 enthalten Symbole die nicht gefüllt sind. Sind offene Symbole füllbar ist der Symbol Wert +100 anwendbar. Im Tag-Namen enthalten die halbgefüllten Symbole die End Kennung *_hf* und die gefüllten Symbole die End Kennung *_f*. In der Bilschirmausgabe werden die halb gefüllten Symbole nicht vollständig dargestellt.

Kommando:	**Wert:**	**Symbol:**
plot_symbols.plus_sign	(1) 101	+
plot_symbols.cross_plus	(2) 102	✳
plot_symbols.point	(3) 103	
plot_symbols.diamond	(4) 104	◇
plot_symbols.triangle	(5) 105	△
plot_symbols.square	(6) 106	□
plot_symbols.cross	(7) 107	×
plot_symbols.circle	108	○
plot_symbols.upside_down_triangle	109	▽
plot_symbols.pointed_5_star	110	✯
plot_symbols.downward_arrow	111	↓
plot_symbols.upward_arrow	112	↑
plot_symbols.left_pointing_arrow	113	←
plot_symbols.right_pointing_arrow	114	→
plot_symbols.star	115	☆
plot_symbols.circle_0	116	⊖
plot_symbols.circle_45	117	⊘
plot_symbols.circle_90	118	⊕
plot_symbols.circle_135	119	⊖
plot_symbols.circle_180	120	⊕
plot_symbols.square_270	121	⊟
plot_symbols.diamond_f	204	◆
plot_symbols.triangle_f	205	▲
plot_symbols.square_f	206	■
plot_symbols.circle_f	208	●
plot_symbols.upside_down_triangle_f	209	▼
plot_symbols.star_f	215	★
plot_symbols.circle_0_hf	216	◒
plot_symbols.circle_45_hf	217	◪
plot_symbols.circle_90_hf	218	◑
plot_symbols.circle_135_hf	219	◓
plot_symbols.circle_180_hf	220	◐
plot_symbols.square_270_hf	221	▬

Tabelle 5.8: PLOT Umgebung: Die Symbole

5.2.7 Kommandos für die X-Achsen

Diese Kommandos werden von `plotprobability`, `plotxy`, `plotxy_2d`, `plot2d`, `plotcell` und `plot3d` verwendet.

Kommando:	Voreinstellung:	Bedeutung:
plot.xtitle [25]	"	Die X-Achsen Beschriftung
plot.xrange [25,25]	[-999.d,-999.d]	-999.d bedeutet, dass der Range in Folge des Wertebereiches definiert wird (+- 5%)
plot.xtype [25]	0	X-Achsen-Typ: 0 lineare Achse, 1 logarithmische Achse, außer x ist die Zeitachse
plot.xaxis	0	Bestimmt wie Achsen verwendet werden: 0 untere Achse ist Bezugsachse, obere Achse wird nicht beschriftet 1 untere Achse ist Bezugsachse; obere Achse wird nicht gezeichnet 2 obere Achse ist Bezugsachse; untere Achse wird nicht gezeichnet
plot.xgrid	0	X-Gitter-Linien: 0 Aus 1 Ein
plot.xgridstyle	0	Art der Linie, die für das Grid verwendet werden soll
plot.xtickv [25,25]	[-999]	enthält die Werte, an denen Hauptticks erscheinen sollen
plot.xtickname [25,25]	"	enthält die Namen, die an den Hauptticks stehen sollen
plot.xtickformat	"	Format, mit dem die Werte an den Tics formatiert werden
plot.xminor [25]	0	enthält die Anzahl der Zwischenticks (0: IDL wählt selbst)
plot.xticks	0	enthält die gewünschte Anzahl der Hauptticks, kann aber von IDL in Zeitreihenplots noch +-1 geändert werden; 0: automatische Generierung
plot.x_use_units [25,25]	"	Bei Verwendung von file=file und x_use_units werden die Daten in die neue Einheit umgerechnet

Tabelle 5.9: PLOT Umgebung: Kommandos für die X-Achsen

5.2.8 Kommandos für die Y-Achsen

Diese Kommandos werden von `plotprobability`, `plotxy`, `plotxy_2d`, `plot2d`, `plotcell` und `plot3d` verwendet.

Kommando:	Voreinstellung:	Bedeutung:
plot.ytitle [25]	"	Die Y-Achsen Beschriftung
plot.yrange [25,25]	[-999.d,-999.d]	-999.d bedeutet das der Range in Folge des Wertebereiches definiert wird
plot.ytype [25]	0	Y-Achsen-Typ: 0 lineare Achse, 1 logarithmische Achse
plot.yaxis	0	Bestimmt wie Achsen verwendet werden. 0 linke Achse ist Bezugsachse und rechte Achse wird nicht beschriftet 1 linke Achse ist Bezugsachse; rechte Achse wird nicht gezeichnet 2 rechte Achse ist Bezugsachse; linke Achse wird nicht gezeichnet.
plot.ygrid	0	Y-Gitter-Linien: 0 Aus 1 Ein
plot.ygridstyle	0	Art der Linie die für den Grid verwendet werden soll
plot.ytickv [25,25]	[-999]	enthält die Werte, an denen Hauptticks erscheinen sollen
plot.ytickname [25,25]	"	enthält die Texte, die an den Hauptticks stehen sollen
plot.ytickformat	"	Format mit dem die Werte an den Tics formatiert werden
plot.yminor [25]	0	enthält die Anzahl der Zwischenticks (0: IDL wählt selbst)
plot.yticks	0	enthält die gewünschte Anzahl der Hauptticks; 0: automatische Generierung
plot.y_use_units [25,25]	"	Bei Verwendung von file=file und y_use_units werden die Daten in die neue Einheit umgerechnet

Tabelle 5.10: PLOT Umgebung: Kommandos für die Y-Achsen

5.2.9 Kommandos für den Titel über den Achsen

Diese Kommandos werden von `plotprobability`, `plotxy`, `plotxy_2d`, `plot2d`, `plotcell` und `plot3d` verwendet.

Kommando:	Voreinstellung:	Bedeutung:
plot.title	"	Der Titel der über den einzelnen Plot geschrieben wird

Tabelle 5.11: PLOT Umgebung: Kommandos für den Titel über den Achsen

5.2.10 Kommandos für die Zeit-Achsen

Diese Kommandos werden von `plotxy`, `plotxy_2d`, `plot2d`, `plotcell` und `plot3d` verwendet.

Kommando:	Voreinstellung:	Bedeutung:
plot.timeformat	"	Ausgabe-Format für Zeit-Label: " bedeutet Voreinstellung auf '!h$:m$:s$@!d$.nn$.Y$' Y$ 4 stellige Jahreszahl; y$ 2 stellige Jahreszahl; N$ kompletter Monatsname; nn$ Nummer des Monats; n$ 3 Anfagsbuchstaben des Monats; d$ Tag des Monats; 0d$ Tag des Monats mit vorangestellter 0 falls benötigt; W$ kompletter Wochentagsname; w$ 3 Anfangsbuchstaben den Wochentages; h$ Stunde; m$ Minute; s$ Sekunde. f$ Sekunden als Bruch. I$ Zeit Intervall in Tagen mit 2 dezimal Stellen. i$ Zeit Intervall in Tagen als ganze Zahle. H$ Zeit Intervall in Stunden als ganze Zahl. @ Carriage Return; ! Line feed.
plot.xgap	31536000.	Bei Zeit-Serien Distanz in Sekunden zwischen zwei Werten, die nicht mehr durch eine Linie miteinander verbunden werden sollen.

Tabelle 5.12: PLOT Umgebung: Kommandos für die Zeit-Achsen

5.2.11 Kommandos für die Legende

Diese Kommandos werden durch die Module `plotxy` und `plotprobability` verwendet.

Kommando:	Voreinstellung:	Bedeutung:
plot.legend_name	"	Legenden-Text der nächsten Datenreihe (sonst wird plot.ytitle verwendet)
plot.legend_title	"	Titel der gesamten Legende
plot.legend_position	0	vordefinierte Positionen: 0 rechts oben neben der Grafik 1 in der linken oberen Ecke des Graphen 2 in der rechten oberen Ecke des Graphen 3 in der rechten unteren Ecke des Graphen 4 in der linken unteren Ecke des Graphen
plot.legend_charsize	1.0	Schriftgrösse der Beschriftungen in der Legende
plot.legend_title_color	schwarz	Farbe des Legendentitels
plot.legend_linewidth	1.	Faktor der Länge der Linien-Symbole
plot.legend_frameposition	[-999,-999]	rel. Legendenposition innerhalb des Plotbereichs
plot.legend_frame_thick	1.	Dicke des Rahmes
plot.legend_frame_color	schwarz	Farbe des Rahmes
plot.legend_frame_fill_color	-999	Füllfarbe des Rahmens -999 Transparent, 0 bis 20 Füll-Farbe des Rahmes
plot.legend_text_color_from_symbol	0	Textfarbenwahl für den Text in der Legende: 0 der Legendentext ist schwarz 1 der Legendentext erhält die Farbe des jeweiligen Symbols
plot.legend_align	[-999,-999]	Ausrichtung des Legendenrahmens
plot.legend		intern: Array[25, 6] enthält alle Titel, Symbole etc.

Tabelle 5.13: PLOT Umgebung: Kommandos für die Legende

5.2.12 Kommandos für den Farbbalken

Diese Kommandos werden durch die Module `plotxy_2d`, `plot2d`, `plot_cell` und `plot3d` verwendet.

Kommando:	Voreinstellung:	Bedeutung:
plot.cbar	1	Farb-Balken: 0 Aus 1 Ein
plot.cbar_title	''	setzt einen Titel über den Farbbalken
plot.cbar_labels	[0..59.] 1	Label des Farb-Balkens: 0 Aus 1 Ein
plot.cbar_labels_step	[0,1]	[n,m] n Startindex des ersten labels m Schritt bis zum nächsten label
plot.cbar_show_maxvalue	1	Ausgabe des Daten-Maximums als Beschriftung am obersten Farbkästchen: 0 Aus 1 Ein
plot.cbar_show_minvalue	1	Ausgabe des Daten-Minimums als Beschriftung am untersten Farbkästchen: 0 Aus 1 Ein
plot.cbar_labels_format	'auto'	Zahlen-Format der Labels: 'auto' maximales Format, wird automatisch bestimmt für die Labels 'X.F' Symbolische Formatierung
plot.cbar_labels_align	'right' ('r')	Ausrichtung der Labels: 'right' rechtsbündig 'center' zentriert 'left' linksbündig
plot.cbar_spacing	0.05	Abstand der Legende zur Achse
plot.cbar_width	0.03	Dicke der Legende
plot.cbar_position	'right' ('r')	setzt die Legende an die definierte Position optimal ist derzeit nur 'right'
plot.cbar_colors		Nur Rückgabe: Enthält die Indizes der Farben die in der Legende verwendet werden
plot.cbar_values		Nur Rückgabe: Enthält die formatierten Werte, die an die Legende geschrieben werden.

Tabelle 5.14: PLOT Umgebung: Kommandos für den Farbbalken

5.2.13 Kommandos für die Fehlerbalken

Diese Kommandos werden durch das Modul `plotxy` verwendet.

Kommando:	Voreinstellung:	Bedeutung:
plot.error_bar.s_hat	0	0 nur senkrechte Fehlerbalken, 1 die Fehlerbalken werden mit einem Strich abgegrenzt
plot.error_bar.s_hat_length	-999.	Länge des Dachs am Fehlerbalken: -999. !D.X_VSIZE/100., n.n gewünschte andere Länge
plot.error_bar.s_thick	-999.	Dicke der Fehlerbalkenlinie: -999. !P.thick, n.n gewünschte andere Dicke
plot.error_bar.s_linestyle	0	Die Art der Linie.
plot.error_bar.s_color	-999.	Farbe des Fehlerbalkens: -999. bedeutet gleiche Farbe wie das Plotsymbol, 0 .. 20 Farbnummer
plot.error_bar.s_nskip	1	Anzahl der Fehlerbalken, die übersprungen werden sollen
plot.error_bar.s_background	1	Fehlerbalken im Vordergrund: 0 Aus 1 Ein

Tabelle 5.15: PLOT Umgebung: Kommandos für die Fehlerbalken

5.2.14 Kommandos für das Gridden von Daten

Diese Kommandos werden durch das Modul `plot2d` verwendet.

Kommando:	Voreinstellung:	Bedeutung:
plot.map_set.triangulate.degree	1	0 Koordinaten der Daten sind im Bogenmaß, 1 Koordinaten der Daten liegen in Grad vor
plot.map_set.trigrid.gs	[0.5,0.5]	Ist ein Vektor mit 2 Elementen [XS,YS], wobei XS der horizontale Abstand zwischen den Grid-Daten-Punkten und YS der vertikale Abstand ist.

Tabelle 5.16: PLOT Umgebung: Kommandos für das Gridden von Daten

5.2.15 Kommandos für die Kartenprojektionen

Diese Kommandos werden durch das Modul `plotmap` verwendet.

Kommando:	Voreinstellung:	Bedeutung:
plot.map_set.projection	'Cylindrical'	Bezeichner der Projektion: CYLINDRICAL, MERCATOR, ORTHOGRAPHIC, SATELLITE, STEREOGRAPHIC, u.a.
plot.map_set.p0lat	-999. 0	Der Breitengrad Punkt der in der Mitte liegen soll (-90 bis 90)
plot.map_set.p0lon	-999. 0	Der Längengrad Punkt der in der Mitte liegen soll (-180 bis 180)
plot.map_set.rot	-999. 0	In Grad die Rotation (0 bis 360)
plot.map_set.isotropic	0	unterschiedlich grosse Abstände, in X und Y Richtung: 0 Aus 1 Ein
plot.map_set.central_azimuth	0	Azimutwinkel in Grad von Nordosten
plot.map_set.ellipsod	[6378206.4, $ 0.00676866, $ 0.9996]	Definiert die Ellipse für die 'Transverse Mercator' oder 'Lambert Conic' Projektion.
plot.map_set.noborder	1	Umrandung der Projektion: 0 Aus 1 Ein
plot.map_set.clip	1	Abschneiden der überstehenden Kartenteile der Projektion: 0 Aus 1 Ein
plot.map_set.grid	1	Gitternetz: 0 Aus 1 Ein
plot.map_set.label	1	Abstand der Beschriftungen
plot.map_set.axis_label	0	Beschriftung an den Achsen: 0 Aus 1 Ein
plot.map_set.continents	1	Zeige Kontinente: 0 Aus 1 Ein
plot.map_set.fill_continents	0	Fülle Kontinente: 0 Aus 1 Ein
plot.map_set.continents_color	schwarz	Farbe für das Füllen
plot.map_set.hires	0	Auflösung der Kartenprojektionen: 0 niedrig, 1 hochaufgelöst
plot.map_set.limit	[-90.,$ -180.,$ 90., $ 180.]	Bereich des Kartenausschnitts der Projektion [Latmin, Lonmin, Latmax, Lonmax]

Tabelle 5.17: PLOT Umgebung: Kommandos für die Kartenprojektionen

5.2.16 Kommandos für die 2D Darstellung

Diese Kommandos werden durch die Module `plot2d, plotcell, plotxy_2d` und `plot3d` verwendet.

Kommando:	Voreinstellung:	Bedeutung:
plot.levels	[0..59.]	Die Level, die für das Contouring herangezogen werden sollen. Mit der Routine defp2d in ein beliebiges anderes Feld änderbar
plot.nlevels	29	Anzahl der gleichverteilten Level über den gesamten Wertebereich. 0 bedeutet, dass definierte Level verwendet werden. (0 bis 32)
plot.min_value	-999	-999 bedeutet das automatische Errechnen des min(Z); ansonsten kann man eine untere Grenze der darzustellenden Z-Werte angeben
plot.max_value	-999	-999 bedeutet das automatische Errechnen des max(Z); ansonsten kann man eine obere Grenze der darzustellenden Z-Werte angeben
plot.fill	1.	gefüllte Contourflächen: 0 Aus, nur Contourlinien , 1 Ein
plot.irregular	0	reguläre Daten: 0 Die Daten liegen auf einem regelmässigem Gitter (2-dim Feld) vor. 1 Ein, bedeutet x, y und z sind Vektoren und werden mittels triangulate auf ein Gitter gebracht. 2 verwendet wird eine sphärische Triangulation, mit trigrid werden reguläre Daten erzeugt 3 verwendet wird eine sphärische Triangulation
plot.closed	0	Mit 1 werden contourlinien über den Rand entlang verbunden.
plot.follow	0	*Seit IDL 5, ist diese Option veraltet*
plot.downhill	0	Mit der Änderung auf 1 werden die Contourlinien vom Berg nach unten mit rechtwinklig abstehenden ticks markiert.
plot.color_table	[0..59]	Damit kann man einen bestimmten Farbindex erzwingen: plot.c_color [0..59] mit -999. Änderbar in einen beliebigen Farbindex, z.B.: plot.c_color[0]=plot.color_nc.red.
plot.max_colors	59	maximale zu verwendende Farbindizes, die über den Farbbereich gleich verteilt werden. Wenn es nicht gesetzt wird und plot.nlevels oder plot.levels verwendet wird, wird plot.max_colors aus diesen ausgerechnet. Mit dem Setzen von plot.max_colors wird der errechnete plot.max_colors überschrieben. (0 bis 59)
plot.zrange	[-999.d,-999.d]	-999.d bedeutet, dass der Range in Folge des Wertebereiches definiert wird
plot.active_color_scheme	0	Damit kann man zwischen mehreren installierten Farbsystemen wechseln. Es stellt die Index Nr des geladen Farbsystems dar.

Tabelle 5.18: PLOT Umgebung: Kommandos für die 2D Darstellung

Kommandos für die Contourlinien

Diese Kommandos werden durch das Modul `plot2d` verwendet.

Kommando:	Voreinstellung:	Bedeutung:
plot.c_annotation	[0..59]="	Alias für das normale Label
plot.c_labels	[0..59]=1	0 kein Label 1 Label
plot.c_charsize	1.	Schriftgrösse der Label
plot.c_linestyle	[0..59] 0	Linientyp
plot.c_thick	[0..59] 1.	die Liniendicke der Linie
plot.c_marker	0	Markierung benutzter Daten: 0 Aus 1 Ein
plot.c_orientation	-999	-999 bedeutet Voreinstellung auf 360. Winkel der Schriftdarstellung (0 bis 360)
plot.c_spacing	-999	-999 bedeutet Voreinstellung auf 0. Abstand der Beschriftung

Tabelle 5.19: PLOT Umgebung: Kommandos für die Contourlinien

5.2.17 Kommandos für die 3D Darstellung

Diese Kommandos werden durch das Modul `plot3d` verwendet.

Kommando:	Voreinstellung:	Bedeutung:
plot.surface.ax	30.	Rotations Winkel der X-Achse aus Sicht des Betrachters in Grad
plot.surface.az	30.	Rotations Winkel der Z-Achse aus Sicht des Betrachters in Grad
plot.surface.bottom	0	Die Farbe die für den Untergrund verwendet werden soll
plot.surface.horizontal	0	Bei gesetztem Flag werden nur Linien des Gitternetzes senkrecht zur Blickrichtung gezeichnet
plot.surface.lego	0	Mit 1 werden Lego-Bausteine ähnliche Säulen dargestellt
plot.surface.skirt	-999.	Z-Wert zur Einzeichnung eines Vorhangs. Voreinstellung ist 'kein Vorhang'. Zur Zeichnung des Vorhangs werden Punkte entlang der Kante der X/Y-Ebene jeweils zum angegebenen Z-Wert verbunden. Zusätzlich werden noch Verbindungen dieser Punkte zu benachbarten Punkten eingezeichnet.
plot.surface.save	0	Mit 1 wird die 3D zu 2D Transformationsmatrix von SURFACE in der Systemvariablen !P.T. gespeichert
plot.surface.ztype	0	Z-Achsen-Typ: 0 lineare Achse, 1 logarithmische Achse
plot.active_color_scheme	0	Damit kann man zwischen mehreren installierten Farbsystemen wechseln. Es stellt die Index Nr des geladen Farbsystems dar.

Tabelle 5.20: PLOT Umgebung: Kommandos für die 3D Darstellung

5.3 Widgets zur Verwendung der PLOT-Struktur

Es gibt eine Reihe von widgets, die die Erstellung einer Grafik mit der PLOT-Struktur vereinfachen. Die Widgets ermöglichen es, per Mausklick bestimmte Einstellungen an der PLOT-Struktur vorzunehmen. Diese Einstellungen werden anschliessend im Quell-Text gespeichert und als Änderung der Parameter der PLOT-Struktur zurückgegeben. Dadurch kann man mit diesen Widgets ein PLOT-Script erstellen, dass dann beim erneuten Durchlauf ohne den Aufruf der Widgets im Batch-Prozess arbeiten kann.

Diese Widgets haben mehrere Steuerworte:

Steuerwort:	**Bedeutung:**
/no_remove	Widget Zeile wird nicht aus dem Quelltext gelöscht, Änderungen an der Struktur werden gespeichert.
/no_comment	Widget Zeile wird aus dem Quelltext gelöscht, Änderungen an der Struktur werden gespeichert. Die Widget Zeile wird aus dem Quelltext gelöscht
/no_replace	Widget Zeile wird nicht aus dem Quelltext gelöscht, Änderungen an der Struktur werden nicht gespeichert.

Tabelle 5.21: PLOT Umgebung: Steuerworte der XP Widgets

Sofern kein Steuerwort übergeben wird, werden Änderungen der Struktur gespeichert und der Widget-Aufruf im Quelltext kommentiert.

Durch die Verwendung eines oder mehrerer Widgets kann z.B. der Quell-Text wie folgt angepasst werden.

```
PRO example1
   restore,get_profile_path()+'example1.sav'
   plotprepare,plot
   plotinit,plot

   ; Widget zur Quelltext Erweiterung
   xp_title,plot
   ; ================================

   plotxy,plot,x=x,y=y1
   plotend,plot
END
```

Beispiel 5.2: PLOT Umgebung: Beispiel mit xp_widget Aufruf

```
PRO example1
   restore,get_profile_path()+'example1.sav'
   plotprepare,plot
   plotinit,plot

   ; Widget zur Quelltext Erweiterung
   plot.title='2000-05-03'
   plot.xtitle='counter'
   plot.ytitle='Voltage [V]'
   ; xp_title,plot ;========= automaticly replaced =========
   ; ================================

   plotxy,plot,x=x,y=y1
   plotend,plot
END
```

Beispiel 5.3: PLOT Umgebung: Script nach Durchlauf des Widgets xp_title

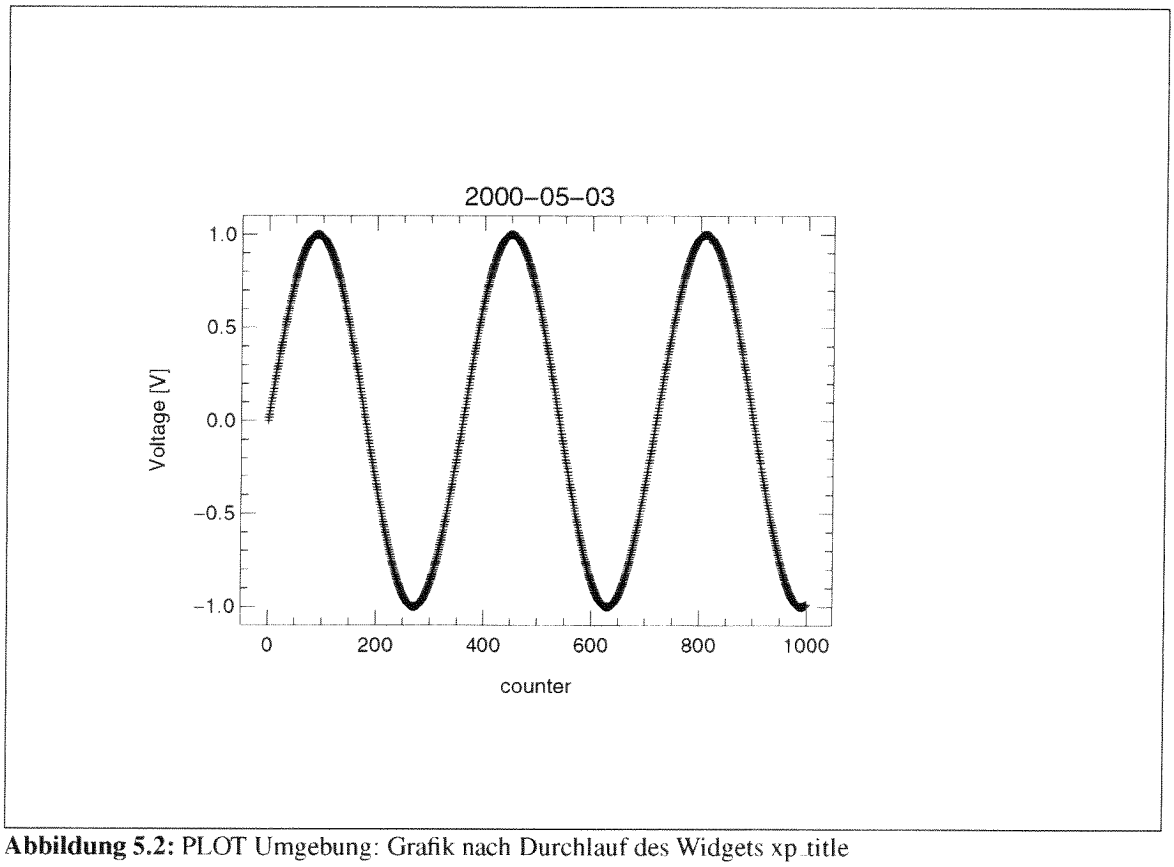


Abbildung 5.2: PLOT Umgebung: Grafik nach Durchlauf des Widgets xp_title

5.3.1 xp_layout

Dieses Widget dient zur Definition des Layouts.

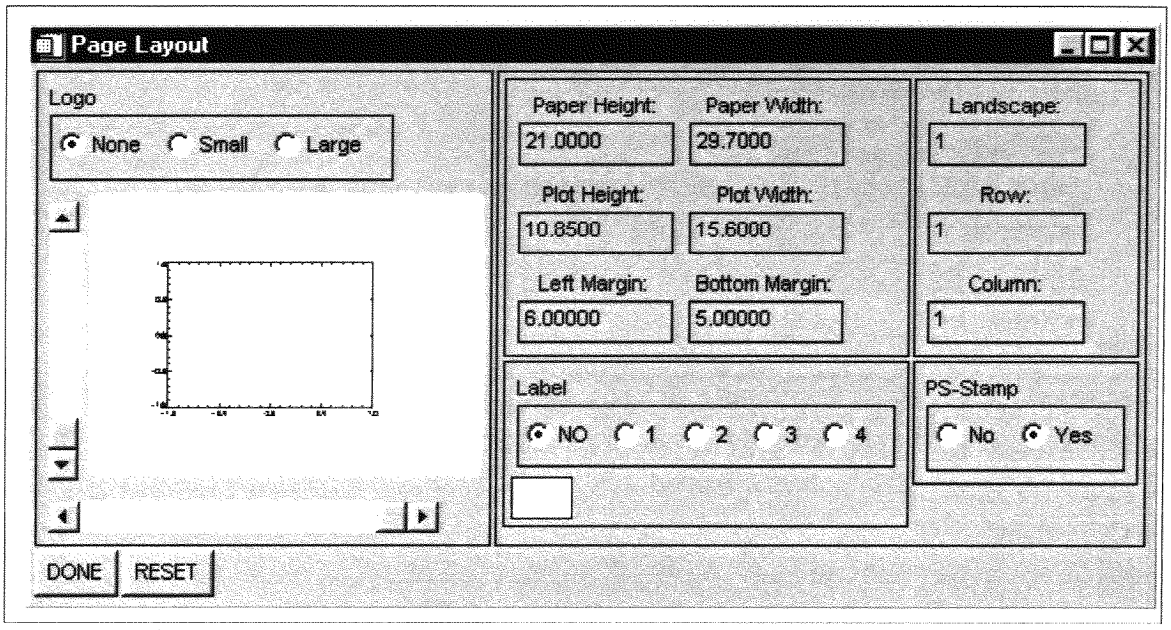


Abbildung 5.3: PLOT Umgebung: xp_layout

Folgende Kommandos können durch das Widget in das Script geschrieben werden (Definition siehe Kapitel 5.2.1 auf Seite 59):

Zusätzliches Kommando:	Beispiel:
plot.paper_height	22.
plot.paper_width	29.
plot.plot_height	14.
plot.plot_width	18.
plot.left_margin	4.
plot.bottom_margin	6.
plot.landscape	0
plot.rows	2
plot.columns	2
plot.logo.type	2
plot.stamp	0
plot.label.name	'a'
plot.label.position	2
plot.x_interstice	90.79
plot.y_interstice	55.21

5.3.2 xp_map_set

Dieses Widget dient zur Definition einer Kartenprojektion.

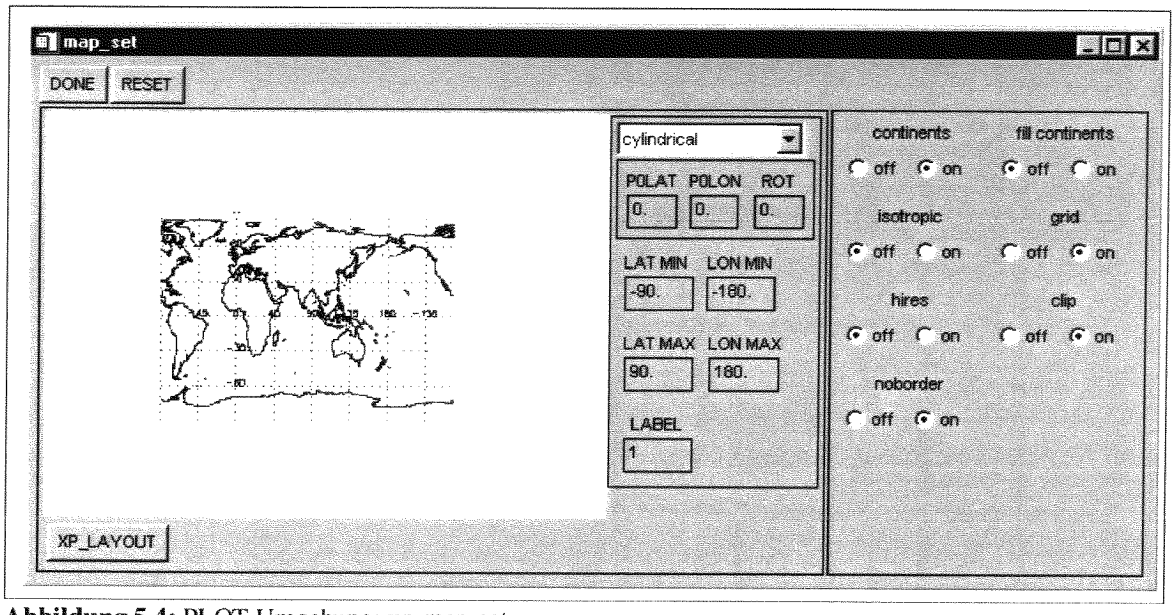


Abbildung 5.4: PLOT Umgebung: xp_map_set

Folgende Kommandos können durch das Widget in das Script geschrieben werden (Definition siehe Kapitel 5.2.15 auf Seite 69):

Zusätzliches Kommando:	**Beispiel:**
plot.map_set.projection	'orthographic'
plot.map_set.p0lat	10.
plot.map_set.p0lon	50.
plot.map_set.rot	10.
plot.map_set.limit	[-40.,-120.,80.,170.]
plot.map_set.continents	0
plot.map_set.fill_continents	1
plot.map_set.grid	0
plot.map_set.label	2
plot.map_set.noborder	0
plot.map_set.hires	1
plot.map_set.clip	0

5.3.3 xp symbols

Dieses Widget dient zur Definition des Symbols und der Symbolfarbe für die nächste Grafik.

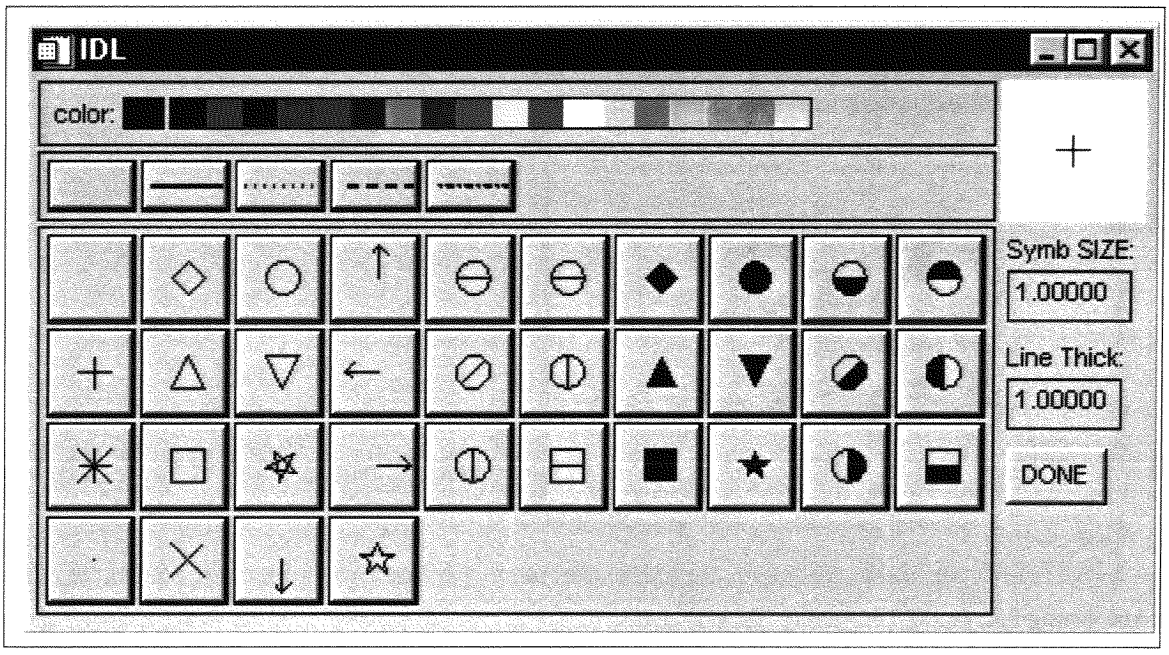


Abbildung 5.5: PLOT Umgebung: xp_symbols.plot

Folgende Kommandos können durch das Widget in das Script geschrieben werden (Definition siehe Kapitel 5.2.6 auf Seite 61):

Zusätzliches Kommando:	**Beispiel:**
plot.psym	-205
plot.color	plot.color_nc.green
plot.symsize	1.5
plot.thick	2.
plot.linestyle	1

5.3.4 xp_axis

Dieses Widget dient der Definition der Achsen für die erste Grafik.

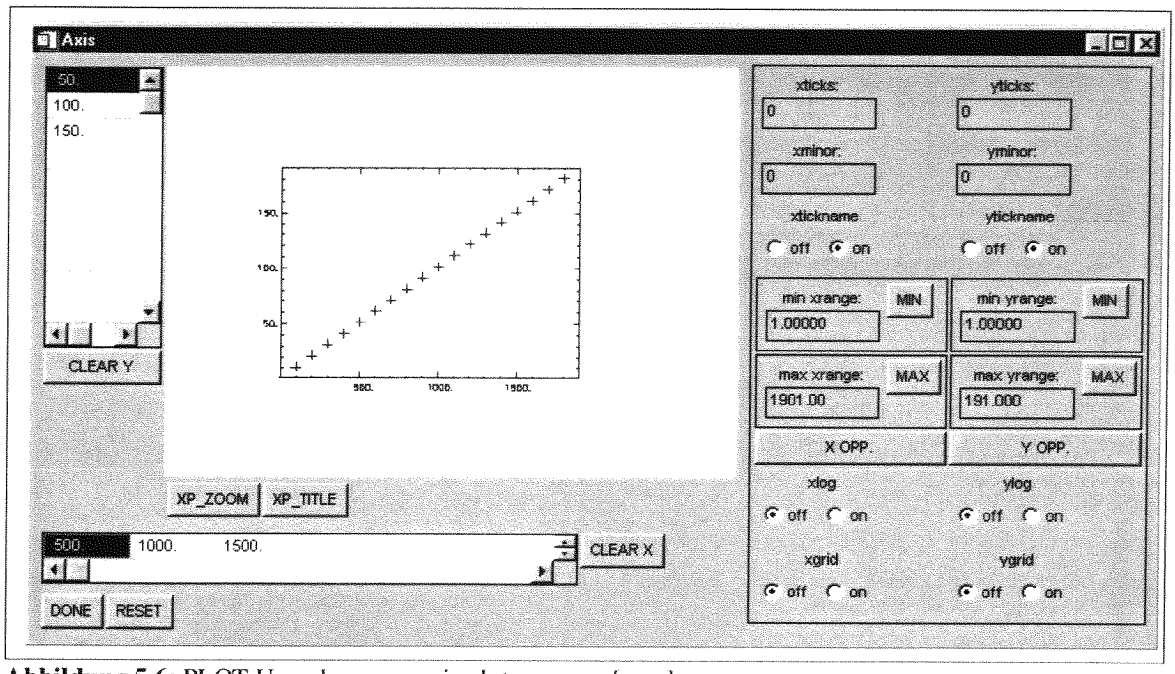


Abbildung 5.6: PLOT Umgebung:xp_axis,plot,x=x,y=y [,z=z]

Folgende Kommandos können durch das Widget in das Script geschrieben werden (Definition siehe Kapitel 5.2.7 auf Seite 63):

Zusätzliches Kommando:	**Beispiel:**
plot.xticks	5
plot.xminor	0
plot.xgrid	1
plot.yticks	5
plot.yminor	10
plot.ygrid	1
plot.xrange[0]	100.
plot.xrange[1]	500.
plot.xtickname[0,0:*]	' '
plot.yrange[0]	0.5
plot.yrange[1]	0.8
plot.ytickname[0,0: 5]	['LOW',' 0.56',' 0.62',' 0.68',' 0.74','MAX']

5.3.5 xp text

Dieses Widget wird verwendet, um einen Text mit dem Mauszeiger in einem Bild zu plazieren und dort dann zu positionieren.

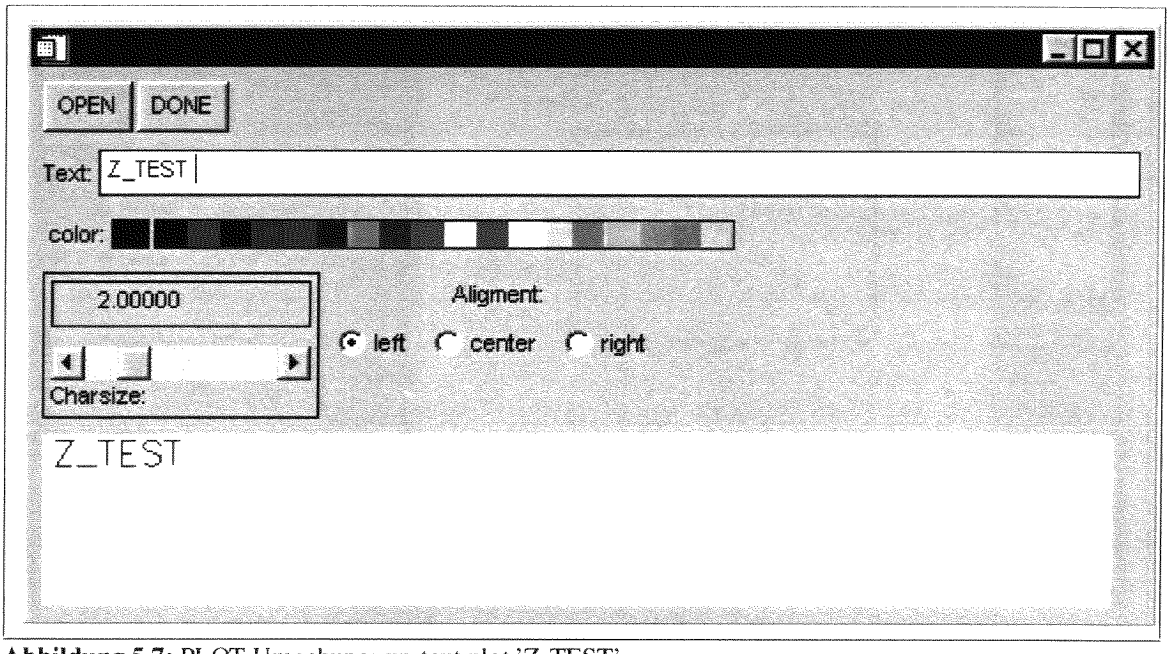


Abbildung 5.7: PLOT Umgebung: xp_text,plot,'Z_TEST'

Folgende Kommandos können durch das Widget in das Script geschrieben werden:

```
xyouts,/norm,0.442254,0.700000,'Z_TEST',color=plot.color_nc.blue, align=0.500000,$
charsize=1.459*plot.charsize_norm_factor
```

5.3.6 xp_title

Dieses Widget dient zur Definition der verschiedenen möglichen Titel einer Grafik.

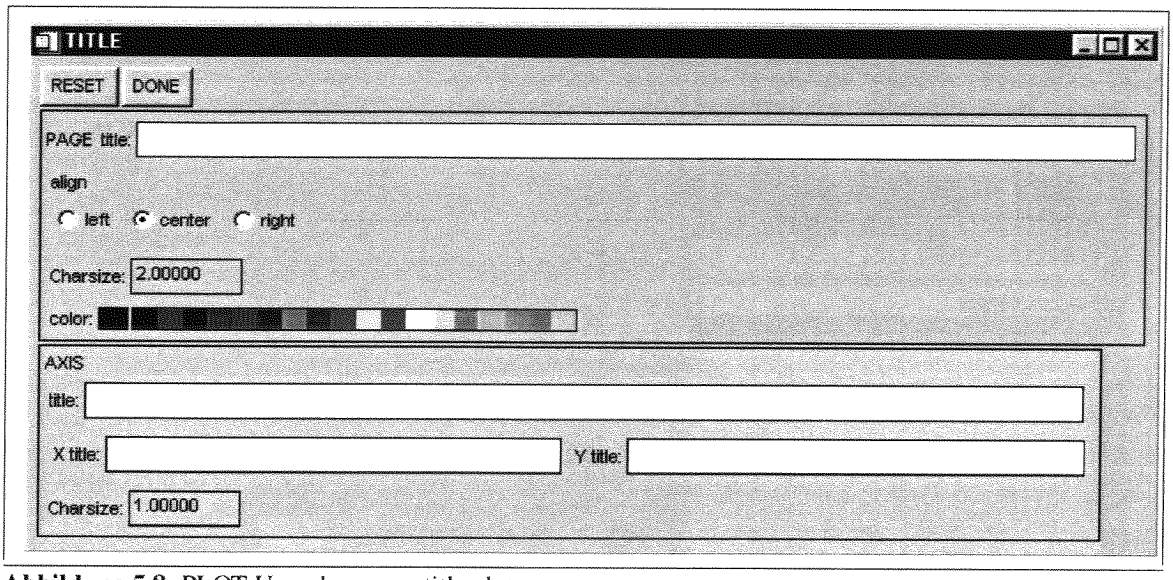


Abbildung 5.8: PLOT Umgebung: xp_title.plot

Folgende Kommandos können durch das Widget in das Script geschrieben werden (Definition siehe Kapitel 5.2.1 auf Seite 59 und Kapitel 5.2.7 auf Seite 63):

Zusätzliches Kommando:	**Beispiel:**
plot.charsize	1.5
plot.title	'Title'
plot.xtitle	'X Title'
plot.ytitle	'Y Title'
plot.page_title	'page_title'
plot.page_title_charsize	4.
plot.page_title_alignment	0.
plot.page_title_color	plot.color_nc.blue

5.3.7 xp_legend

Dieses Widget ermöglicht Manipulationen an der Legende, z.B. die freie Positionierung der Legende mit der Maus in einer Grafik. Die einzelnen Beschriftungen und Eigenschaften des Rahmens können ebenfalls geändert werden.

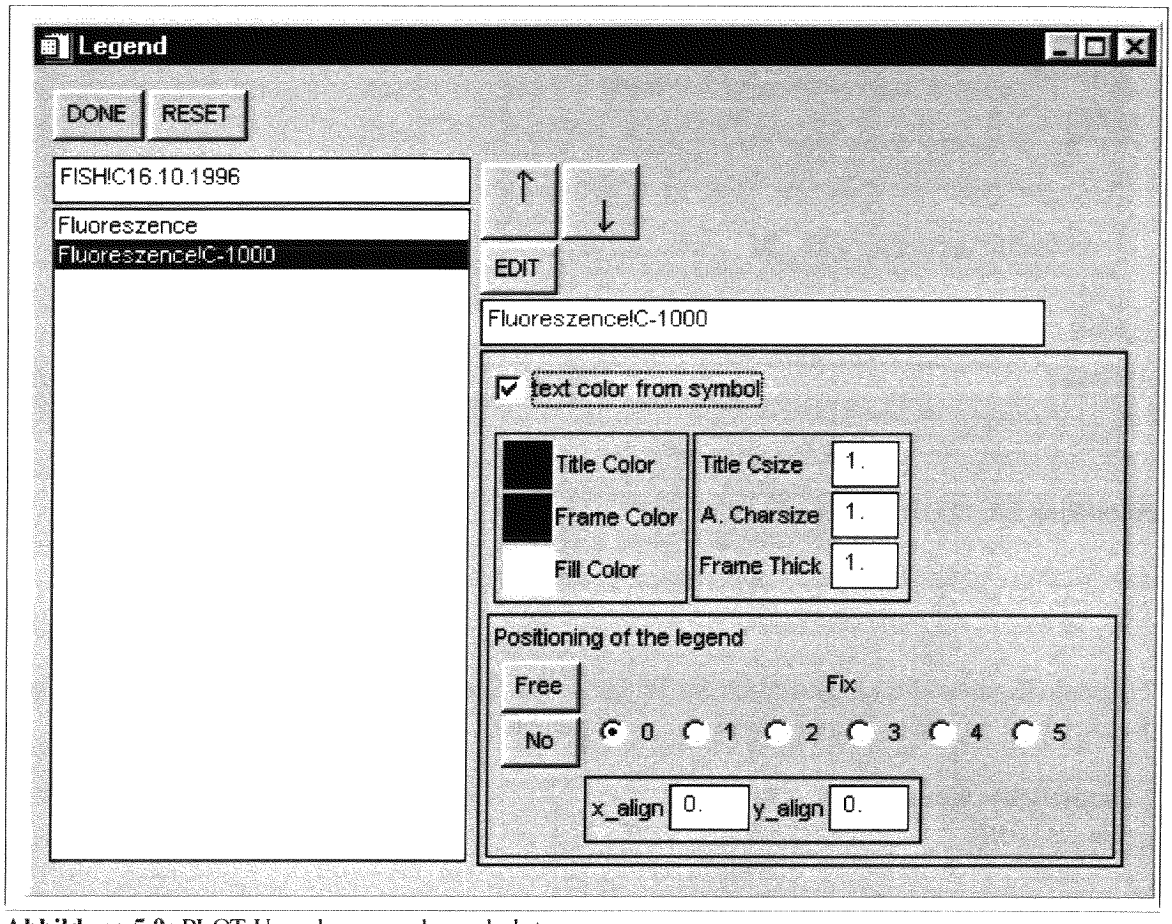


Abbildung 5.9: PLOT Umgebung: xp_legend.plot

Folgende Kommandos können durch das Widget in das Script geschrieben werden (siehe Definition Kapitel 5.2.11 auf Seite 66):

Zusätzliches Kommando:	**Beispiel:**
plot.legend_align	[1.,1.]
plot.legend_frameposition	[0.653115,0.681106]
plot.legend_title	'2000-05-03'
plot.legend_text_color_from_symbol	1
plot.legend_frame_color	plot.color_nc.blue
plot.legend_frame_fill_color	plot.color_nc.light_grey
plot.legend_title_color	plot.color_nc.blue
plot.legend_charsize	1.3
plot.legend_frame_thick	2.
plot.legend[0: 1,*]	legend_reorg(plot,['RES','RES/2'], [1,0])

5.4 Die Module zur Anwendung der PLOT-Struktur

Derzeit verfügen wir über die folgenden Module: `plot_probability`, `plotxy`, `plot_map`, `plot2d`, `plotxy_2d`, `plotcell` und `plot3d`. Anhand einer Reihe von Beispielen werden die verschiedenen Möglichkeiten dieser Module und die Kombination der Module dargestellt. Neben der Bildschirmausgabe ist eine Postscriptausgabe und die Ausgabe in Form eines GIF-Films möglich. In den einzelnen Modulen sind Definitionen getroffen worden, so dass die Bildschirmausgabe nahezu mit der Postscriptausgabe übereinstimmt.

Alle PLOT-Module ausser `plot_map` stellen Daten, die übergeben werden müssen, dar.

Diese Module können Daten aus der ICG-Daten-Struktur, Daten aus einer Datei oder direkt übergebene Daten verarbeiten.

Bei der Übergabe einer ICG-Daten-Struktur mit *icg_struct=icg_struct* oder bei der Übergabe eines Datei-Namens mit *file=file* wird die Eingabe der *short_names* auf der Parametereingabe von *X* und *Y* erwartet. Die drei Aufrufmöglichkeiten lauten wie folgt:

```
plotxy,plot,x='time',y='F12',icg_struct=icg_struct
plotxy,plot,x='time',y='F12',file='test.nc'
plotxy,plot,x=[1,3],y=[100,200]
```

Die Beschreibung befasst sich ausführlich mit der direkten Datenübergabe.

5.4.1 PLOT_PROBABILITY

Mit dem Modul `plot_probability` werden kumulierte Häufigkeitsverteilungen auf sogenanntem Wahrscheinlichkeitspapier erstellt.

5.4.2 PLOTXY

Mit dem Modul `plotxy` werden XY-Graphen erstellt. Das Programm unterstützt bis zu zwei X- und Y-Achsen. Fehlerbalken können symmetrisch oder asymmetrisch dargestellt werden. Eine Legende wird bei der Verwendung von mehr als einer Kurve automatisch dargestellt.

5.4.3 PLOT2D

Mit dem Modul `plot2d` werden contour-Graphen erstellt. Die verwendeten Level werden durch einen Farb-Balken veranschaulicht. Sowohl Linien- als auch Flächen-Contourplots können erstellt werden. Sobald `plot2d` in Verbindung mit Kartenprojektionen verwendet wird, werden gefüllte Contouren mit dem *cell_fill*-Algorithmus erzeugt. Hierbei ist auch zu beachten, dass die Daten aufsteigend angeordnet sein müssen.

5.4.4 PLOTCELL

Das Modul `plotcell` erstellt einen Flächen-Contour-Graphen. Im Unterschied zu `plot2d` werden die Daten nicht interpoliert. Die Werte werden verschiedenen Farben, den Leveln, zugeordnet. Das Bild besteht aus einer Vielzahl miteinander verbundener Rechtecke.

5.4.5 PLOTXY_2D

Mit dem Modul `plotxy_2d` wird ein XY-Graph erstellt. Im Unterschied zu `plotxy` wird die Symbol-Farbe aus der Variablen Z abgeleitet. Die Farbeinteilung lässt sich wie bei `plot2d` steuern. Die verwendeten Level werden durch einen Farb-Balken veranschaulicht.

5.4.6 PLOT3D

Mit dem Modul `plot3d` lassen sich die Daten, die man auch mit `plot2d` darstellen kann, in einer räumlichen Ansicht, in Form eines Drahtmodells darstellen. Bei Übergabe einer vierten Größe (shades=s) wird das Drahtmodell eingefärbt. Damit ist eine 4-D Darstellung möglich.

5.4.7 PLOT_MAP

Mit diesem Modul `plot_map` lässt sich eine Kartenprojektion erstellen und ein Koordinatensystem definieren, auf das sich dann alle weiteren Graphen beziehen. Optional können auch Kontinente in die Karten eingezeichnet werden.

5.4.8 Seitensteuerung

Mit *plot.rows* und *plot.columns* legt man die Anzahl der Bilder in Zeilen und Spalten auf einer Seite fest. Mit *plot.new* wird gesteuert, ob die darauffolgende Grafik ein neues Bild an einer neuen Position erzeugt oder einen Plot mit einem overlay-plot ergänzt. Sobald *plot.new=1* ist, wird für die darauffolgende Grafik ein neues Koordinatensystem an der nächsten Position verwendet. Ist die letzte Position auf der Seite erreicht, wird die nächste Grafik auf der nächsten Seite an der ersten Position erstellt.
Dadurch kann man durch Ändern von *plot.rows* und *plot.columns* die Anzahl der Seiten bestimmen. Die Schriftgrösse und die Längen der Achsen-Tics werden auch durch die Anzahl der Bilder auf einer Seite bestimmt. Bei dem `plotxy-Modul` ist die Anzahl der Legenden-Einträge auf 25 begrenzt. Zum Steuern der Information der Daten, die in die Legende aufgenommen werden sollen, gibt es drei Möglichkeiten:

plot.new=0	nächster plotxy bewirkt einen Eintrag in der Legende
plot.new=2	alle bisherigen Einträge in der Legende werden gelöscht
plot.new=3	nächster plotxy bewirkt keinen Eintrag in der Legende

Die Schalter 2 und 3 sind derzeit nur für `plotxy` definiert.

Mit `plotnew, plot, 1` erreicht man für das `plotxy` Modul, dass die nächste Grafik über die vorhergehende gezeichnet wird (siehe Beispiel 5.5.8 auf Seite 98)
Mit dem Schalter *plot.plot_frameposition* gibt es die Möglichkeit, einen verkleinerten Plot an irgendeine Stelle im eigentlichen Plot zu stellen. Dadurch kann man auf eine besondere Gegebenheit hinweisen. Die Koordinaten beziehen sich auf den Rahmen aus X- und Y-Achse. 0 ist X_min und Y_min und 1 ist X_max und Y_max (siehe Beispiel 5.5.8 auf Seite 98),
Der Aufruf von `set_frame` nach der Definition der Position in *plot.plot_frameposition* ermöglicht das Plazieren jedes anderen Moduls an der gewünschten Stelle (siehe Beispiel 5.5.10).

Der Aufruf `plot_text,plot` bewirkt, dass die Definitionen aus der Text-Umgebung (siehe Kapitel 5.2.4 auf Seite 60) in dem aktuellen Plot ausgeben werden.

5.4.9 Farbsysteme

In unserer PLOT-Umgebung sind die ersten 20 Farben als Konstanten definiert. Der restliche Farbraum lässt sich mit einem oder mehreren Farbsystemen auffüllen. Dies geschieht mit dem Befehl `color_scheme,plot,scheme_code=X`. X stellt hier einen oder mehrere der Farbsystemnamen dar. Um zwischen einzelnen Farbsystemen zu wechseln, wird das Steuerwort *plot.active_color_scheme* (siehe Beispiel 5.21 auf Seite 118) verwendet. Der Wert entspricht dem Index von X. In dieser Weise können gleichzeitig mehrere Farbsysteme verwendet werden.

Derzeit sind folgende Farbsysteme definiert:

ct_blue	
ct_blue_green	
ct_blue_green_yellow_red	
ct_blue_red	
ct_blue_red_yellow	
ct_cyan_green_yellow_red	
ct_grey_191_32	
ct_grey_220_70	
ct_pastel	
ct_yellow_red_blue	
ct_yellow_red_blue_green_black	

Tabelle 5.22: PLOT Umgebung: definierte Farbsysteme

Diese Farbsysteme sind mit dem widget `x_def_colortable` erstellt worden.

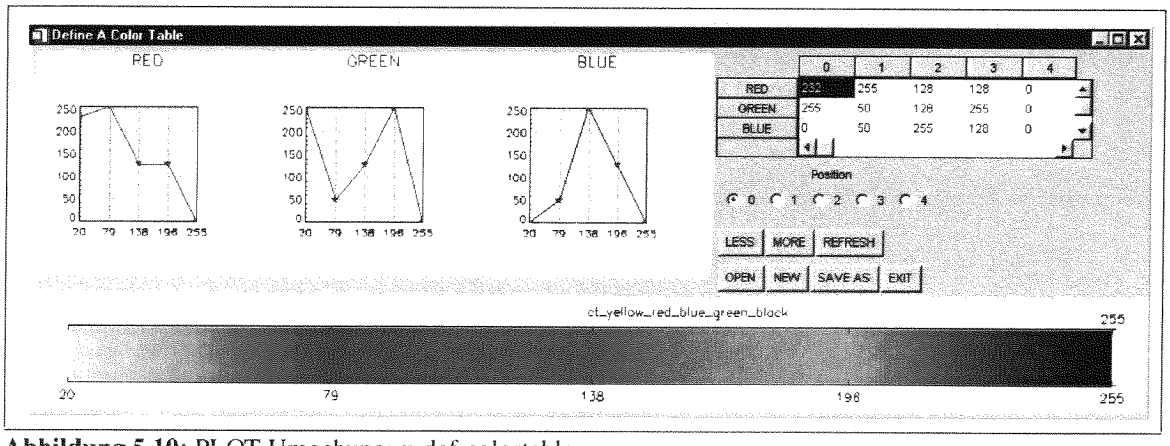


Abbildung 5.10: PLOT Umgebung: x_def_colortable

5.5 Darstellung von Daten

Mit einer Reihe von Beispielen werden Möglichkeiten dieser PLOT-Umgebung vorgestellt.

5.5.1 plot_probability: kumulierte Häufigkeitsverteilung

```
PRO example2
   x=randomn(seed,300)

   plotprepare,plot
   plot.psflag=1
   plot.psfile='example2'
   plotinit,plot
   plot.xgrid=1
   plot.ygrid=1

   plot_probability,plot,x=x

   plotend,plot
END
```

Beispiel 5.4: PLOT Umgebung: plot_probability

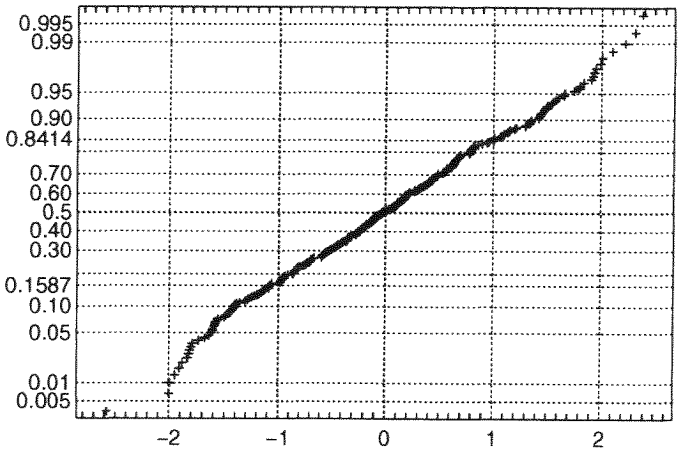


Abbildung 5.11: PLOT Umgebung: plot_probability: kumulierte Häufigkeitsverteilung

5.5.2 plotxy: Darstellung einer einzelnen Datenreihe

```
PRO example3
   RESTORE,get_profile_path()+'example1.sav'

   plotprepare,plot
   plot.psflag=1
   plot.psfile='example3'
   plotinit,plot

   plot.psym  =0
   plot.yrange=[-1.2,1.2]
   plot.color =plot.color_nc.red
   plot.xtitle='TIME'
   plot.ytitle='SIN(x/180*!7p!X)'
   plotxy,plot,x=x,y=y1,/time

   plotend,plot
END
```

Beispiel 5.5: PLOT Umgebung: plotxy: eine Datenreihe

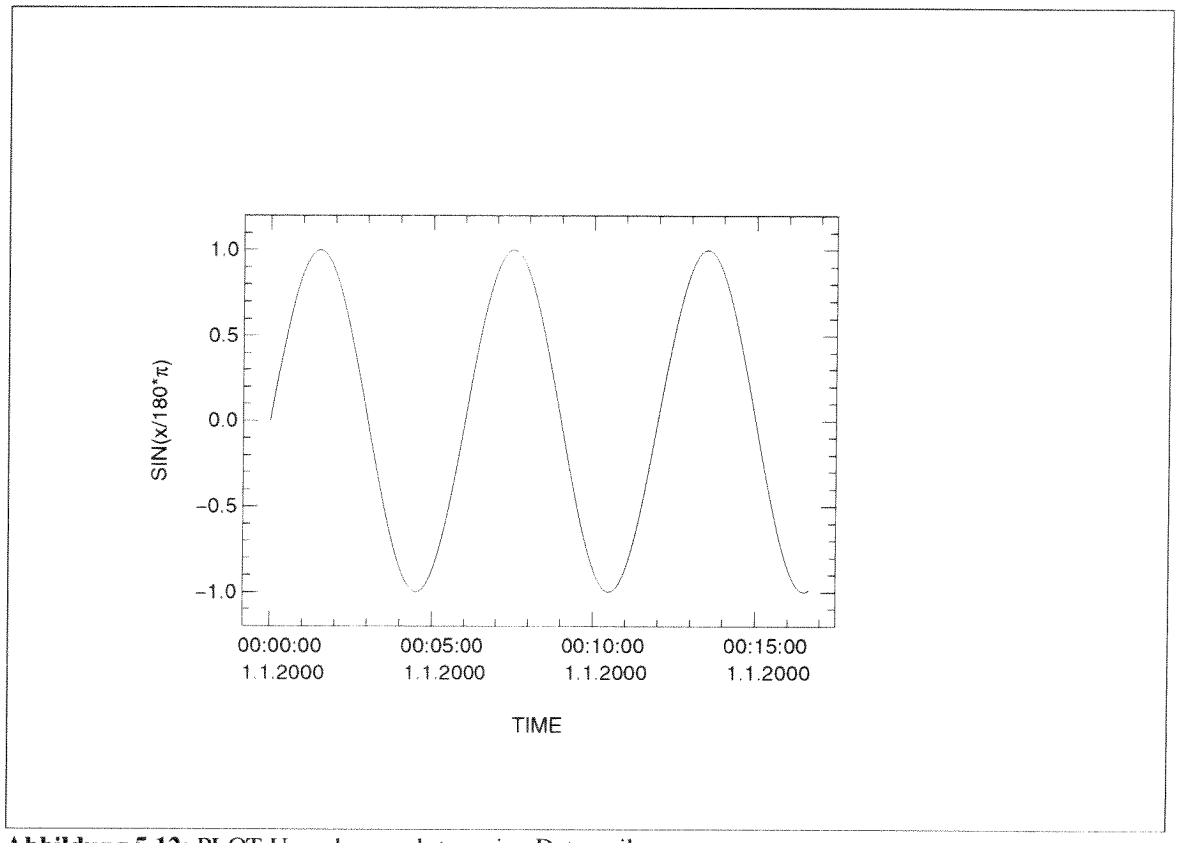


Abbildung 5.12: PLOT Umgebung: plotxy: eine Datenreihe

5.5.3 plotxy: zwei Datenreihen overlay

```
PRO example4
   RESTORE,get_profile_path()+'example1.sav'

   plotprepare,plot
   plot.psflag=1
   plot.psfile='example4'
   plotinit,plot

   plot.psym  =0
   plot.yrange=[-1.2,1.2]
   plot.color =plot.color_nc.red
   plot.xtitle='TIME'
   plot.ytitle='SIN'
   plot.legend_name='SIN(x/180*!7p!X)'
   plotxy,plot,x=x,y=y1,/time

   plot.color =plot.color_nc.blue
   plot.legend_name='SIN(x/180*!7p!X+2)!U2!N'
   plotxy,plot,x=x,y=y2,/time

   plotend,plot
END
```

Beispiel 5.6: PLOT Umgebung: plotxy: zwei Datenreihen overlay

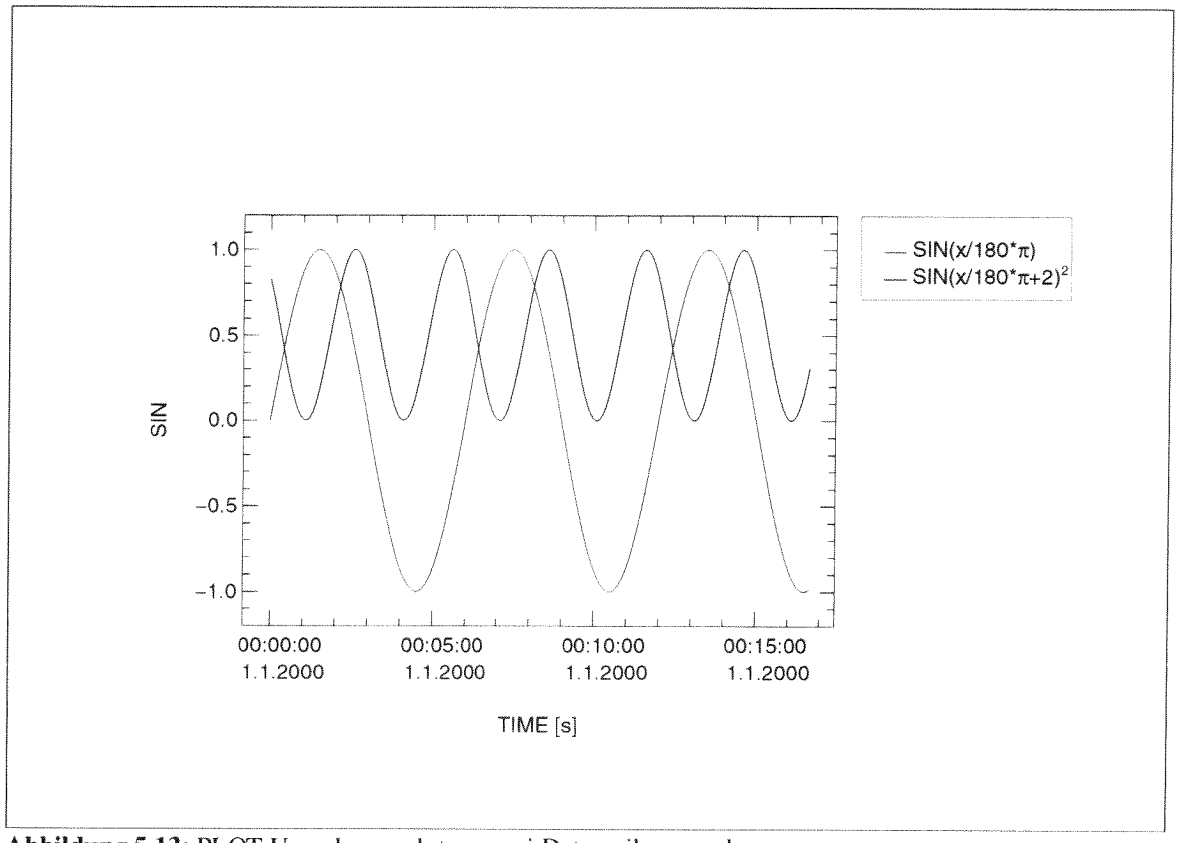


Abbildung 5.13: PLOT Umgebung: plotxy: zwei Datenreihen overlay

5.5.4 plotxy: zwei Datenreihen auf zwei Seiten

```
PRO example5
   RESTORE,get_profile_path()+'example1.sav'

   plotprepare,plot
   plot.psflag=1
   plot.psfile='example5'
   plotinit,plot

   plot.psym  =0
   plot.yrange=[-1.2,1.2]
   plot.color =plot.color_nc.red
   plot.xtitle='TIME'
   plot.ytitle='SIN(x/180*!7p!X)'
   plotxy,plot,x=x,y=y1,/time

   plot.new=1
   plot.color =plot.color_nc.blue
   plot.ytitle='SIN(x/180*!7p!X+2)!U2!N'

   plotxy,plot,x=x,y=y2,/time

   plotend,plot
END
```

Beispiel 5.7: PLOT Umgebung: plotxy: zwei Datenreihen zwei Seiten

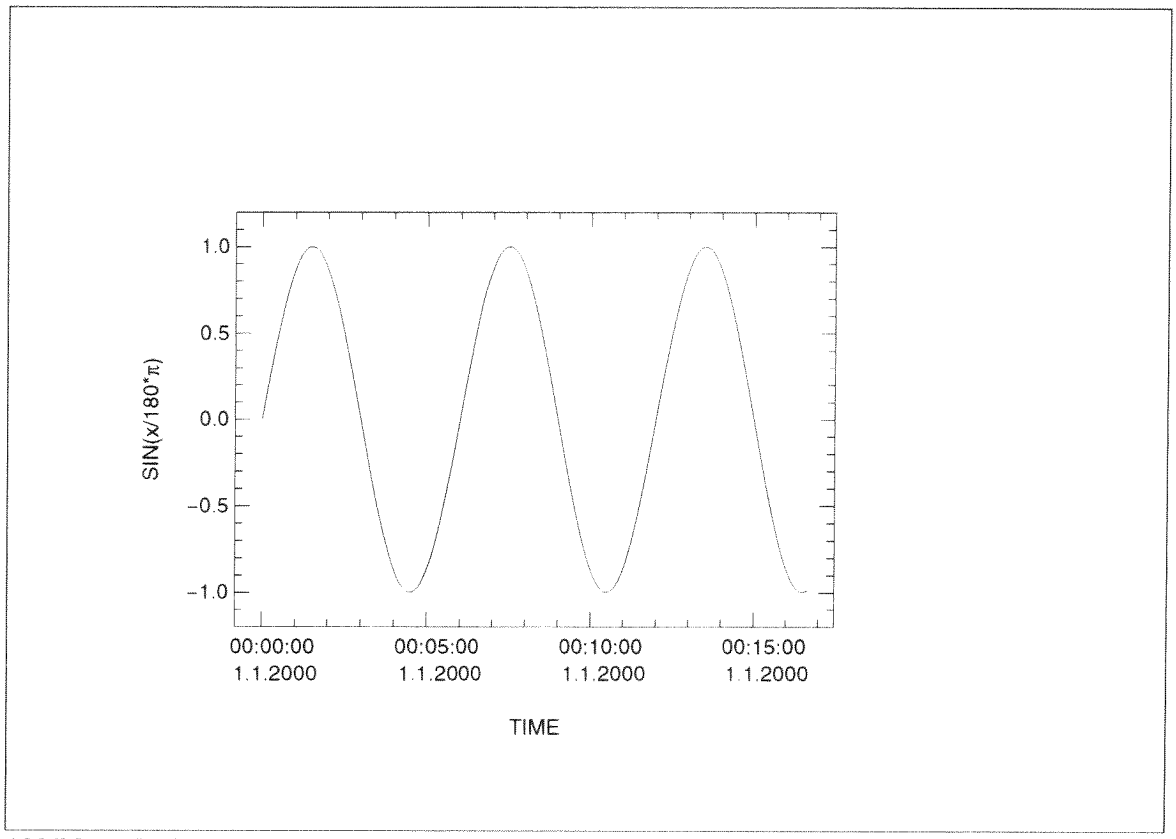


Abbildung 5.14: PLOT Umgebung: plotxy: zwei Datenreihen auf zwei Seiten (Seite: 1)

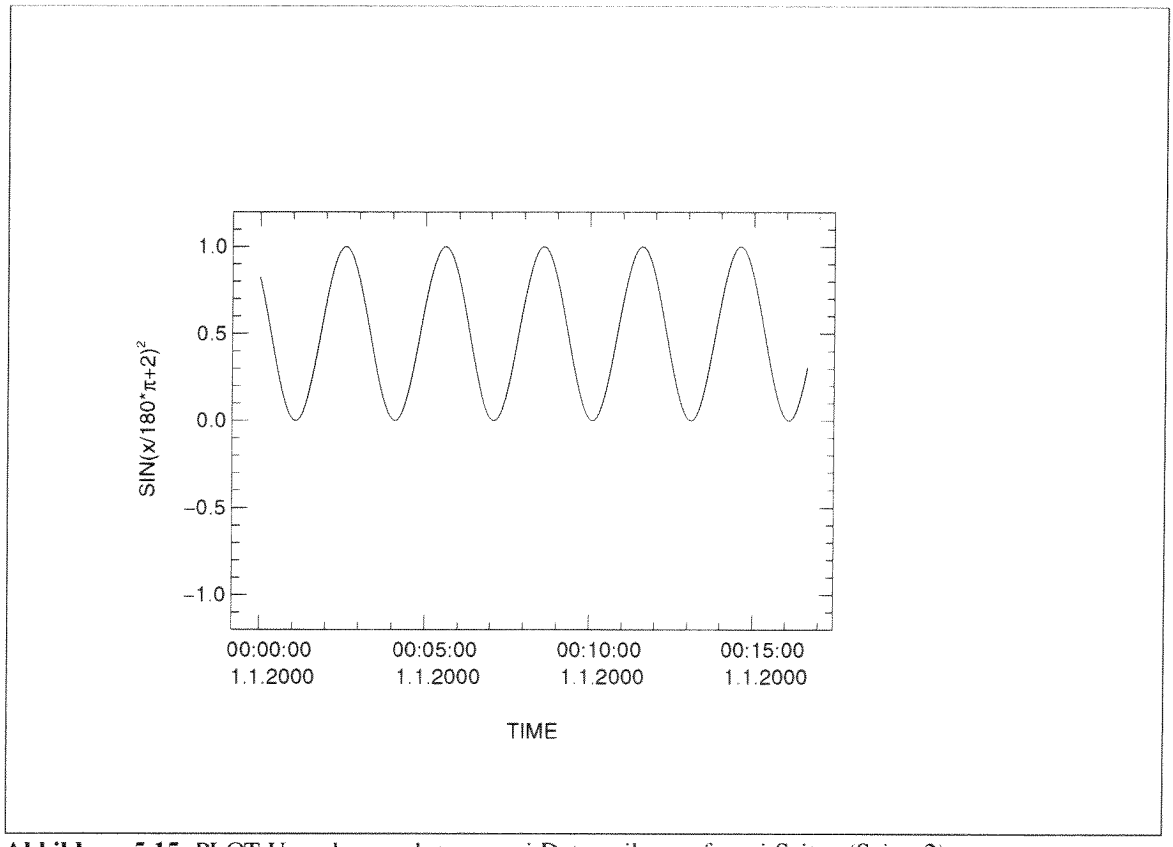


Abbildung 5.15: PLOT Umgebung: plotxy: zwei Datenreihen auf zwei Seiten (Seite: 2)

5.5.5 plotxy: zwei Datenreihen in zwei Bildern auf einer Seite

```
PRO example6a
   RESTORE,get_profile_path()+'example1.sav'

   plotprepare,plot
   plot.psflag=1
   plot.psfile='example6a'
   plot.columns=2
   plot.rows =1
   plotinit,plot

   plot.psym  =0
   plot.yrange=[-1.2,1.2]
   plot.color =plot.color_nc.red
   plot.xtitle='TIME'
   plot.ytitle='SIN'
   plot.legend_name='SIN(x/180*!7p!X)'
   plotxy,plot,x=x,y=y1,/time

   plot.new=1
   plot.color =plot.color_nc.blue
   plot.legend_name='SIN(x/180*!7p!X+2)!U2!N'
   plotxy,plot,x=x,y=y2,/time

   plotend,plot
END
```

Beispiel 5.8: PLOT Umgebung: plotxy: zwei Bilder auf einer Seite

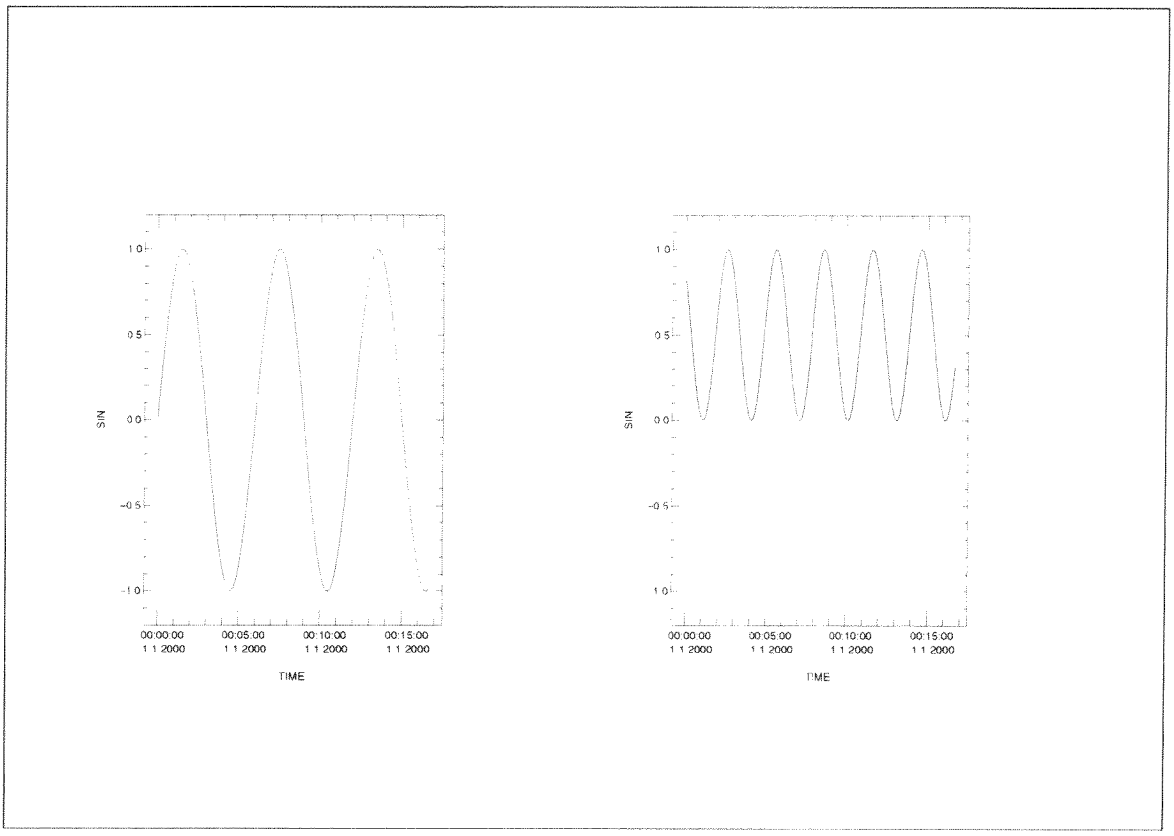


Abbildung 5.16: PLOT Umgebung: plotxy: zwei Bilder auf einer Seite

5.5.6 plotxy: zwei Datenreihen in zwei Bildern matrixartig auf einer Seite nebeneinander angeordnet

```
PRO example6b
   RESTORE,get_profile_path()+'example1.sav'
   plotprepare,plot
   plot.psflag=1
   plot.psfile='example6b'
   plot.columns=2
   plot.rows =1
   plot.x_interstice=0.
   plotinit,plot

   plot.psym  =0
   plot.yrange=[-1.2,1.2]
   plot.color =plot.color_nc.red
   plot.xtitle='TIME'
   plot.ytitle='SIN'
   plot.legend_name='SIN(x/180*!7p!X)'
   plotxy,plot,x=x,y=y1,/time

   plot.new=1
   plot.color =plot.color_nc.blue
   plot.legend_name='SIN(x/180*!7p!X+2)!U2!N'
   plot.ytitle=' '
   plot.ytickname=' '
   plotxy,plot,x=x,y=y2,/time

   plotend,plot
END
```

Beispiel 5.9: PLOT Umgebung: plotxy: Bilder matrixartig nebeneinander

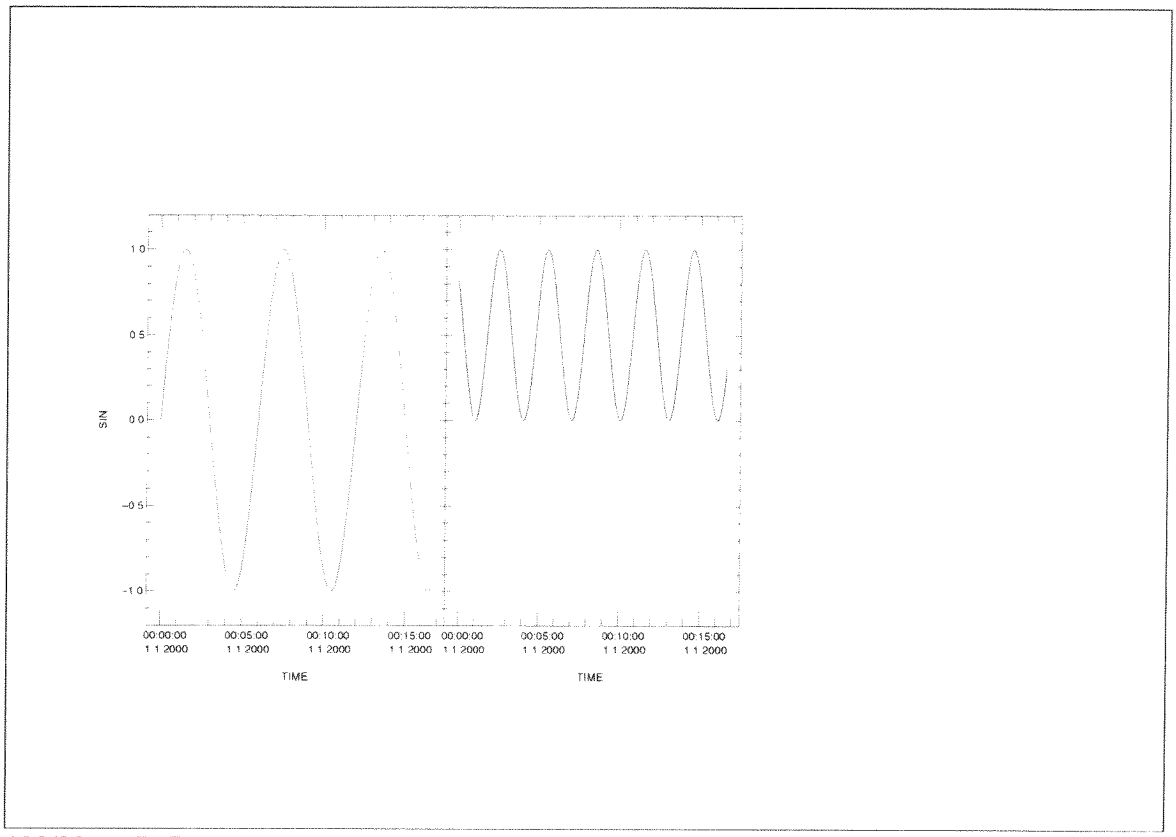


Abbildung 5.17: PLOT Umgebung: plotxy: Anordnung von zwei Bildern matrixartig nebeneinander

5.5.7 plotxy: zwei Datenreihen in zwei Bildern auf einer Seite [[y1],[y2]]

Dieser Mechanismus zeigt die Verwendung von `plotxy` mit mehrdimensionalen Feldern. In dieser Weise werden nur mit `plotxy` in einer Matrix angeordnete Grafiken erstellt. Diese Vereinfachung ist nicht mit allen plot-Kommandos möglich.

```
PRO example6c
    RESTORE,get_profile_path()+'example1.sav'
    plotprepare,plot
    plot.psflag=1
    plot.psfile='example6c'
    plot.x_interstice=0.
    plotinit,plot

    plot.psym  =0
    plot.yrange=[-1.2,1.2]
    plot.color =[plot.color_nc.red,plot.color_nc.blue]
    plot.xtitle='TIME'
    plot.ytitle='SIN'
    plot.legend_name='SIN(x/180*!7p!X)'
    plotxy,plot,x=x,y=[[y1],[y2]],/time

    plotend,plot
 END
```

Beispiel 5.10: PLOT Umgebung: plotxy: Bilder matrixartig mit [[y1],[y2]]

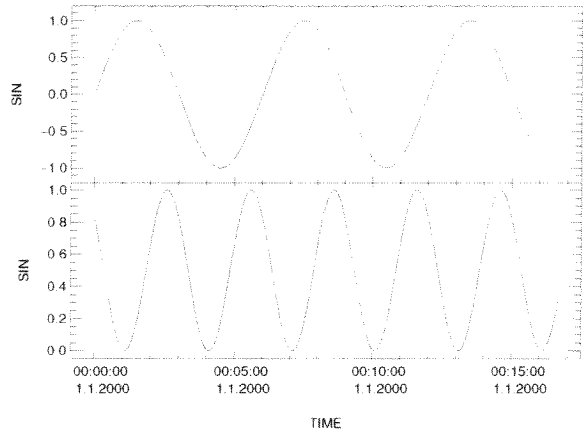


Abbildung 5.18: PLOT Umgebung: plotxy: zwei Bilder matrixartig übereinander auf einer Seite [[y1],[y2]]

5.5.8 plotxy: eingefügtes Bild

```
PRO example7
  RESTORE,get_profile_path()+'example1.sav'

  plotprepare,plot
  plot.psflag=1
  plot.psfile='example7'
  plot.stamp =0
  plotinit,plot

  plot.psym  =0
  plot.yrange=[-1.2,2]
  plot.color =plot.color_nc.red
  plot.xtitle='TIME'

  plot.ytitle='SIN(x/180*!7p!X)'
  plotxy,plot,x=x,y=y1*EXP(-x/800.),/time

   plotnew,plot,1
   plot.frame.position =[0.65,0.65,0.95,0.95]
   plot.charsize=plot.charsize*0.75
   plot.yrange=[0,1]
   plot.xrange=[-1.2,2]
   plot.color =plot.color_nc.blue
   plot.xtitle='SIN(x/180*!7p!X)'
   plot.ytitle='SIN(x/180*!7p!X+2)!U2!N'
   plotxy,plot,x=y1,y=y2


  plotend,plot
END
```

Beispiel 5.11: PLOT Umgebung: plotxy: eingefügtes Bild

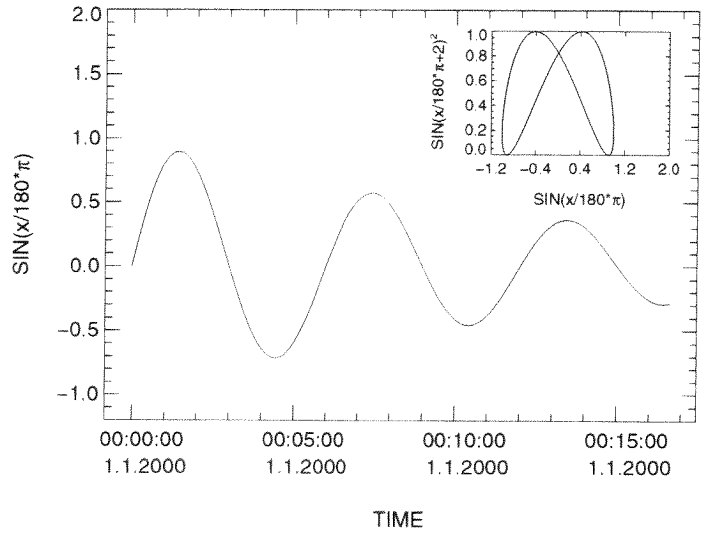


Abbildung 5.19: PLOT Umgebung: plotxy: eingefügtes Bild

5.5.9 plotxy: zwei Y-Achsen

```
PRO example8
   RESTORE,get_profile_path()+'example1.sav'

   plotprepare,plot
   plot.psflag=1
   plot.psfile='example8'
   plotinit,plot

   plot.yaxis =1
   plot.psym  =0
   plot.yrange=[-1.2,1.7]
   plot.color =plot.color_nc.red
   plot.xtitle='TIME'
   plot.ytitle='SIN(x/180*!7p!X)'
   plotxy,plot,x=x,y=y1,/time

   plot.yaxis =2
   plot.yrange=[0,1.7]
   plot.color =plot.color_nc.blue
   plot.ytitle='SIN(x/180*!7p!X+2)!U2!N'
   plot.legend_position=1
   plotxy,plot,x=x,y=y2,/time

   plotend,plot
END
```

Beispiel 5.12: PLOT Umgebung: plotxy: zwei Y Achsen

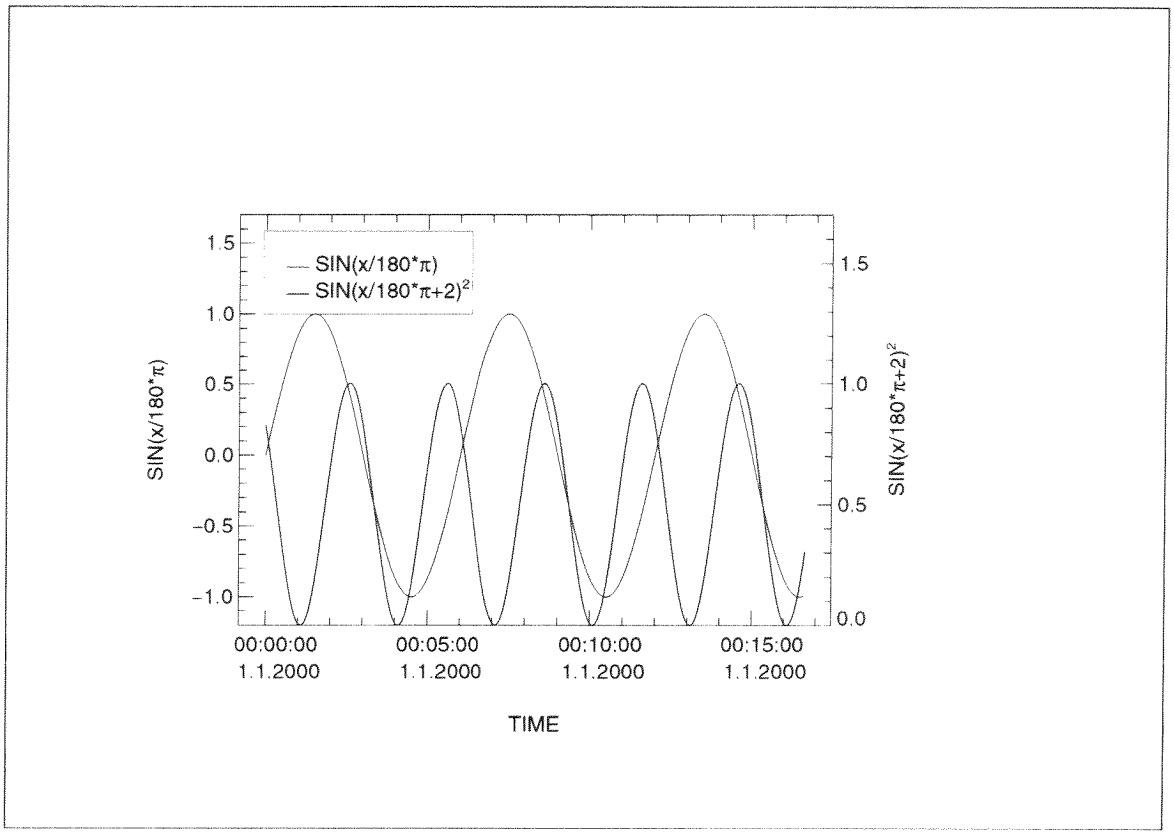


Abbildung 5.20: PLOT Umgebung: plotxy: zwei Y Achsen

5.5.10 plotxy: Markieren von Bildinformationen

```
PRO example9
   RESTORE,get_profile_path()+'example1.sav'

   plotprepare,plot
   plot.psflag=0
   plot.psfile='example9'
   plotinit,plot
   plot.xrange=[string2js(['2000-01-01 00:05:00', $
               '2000-01-01 00:15:00'])]
   set_frame,PLOT,x=x,y=y1,/time,type='XY'

   xv=[400,500] - (string2js(js2string(x[0],format='Y-M-D')))
   yv=[-1,1]*1.1
   POLYFILL,[xv[0],xv[1],xv[1],xv[0]],[yv[0],yv[0],yv[1],yv[1]],$
            /data,color=plot.color_nc.orange


   xv=[700,800d] - (string2js(js2string(x[0],format='Y-M-D')))
   yv=[-1,1]*1.1
   POLYFILL,[xv[0],xv[1],xv[1],xv[0]],[yv[0],yv[0],yv[1],yv[1]],$
            /data,color=plot.color_nc.medium_grey

   plot.psym =0
   plot.yrange=[-1.2,1.2]
   plot.color =plot.color_nc.red
   plot.xtitle='TIME'
   plot.ytitle='SIN(x/180*!7p!X)'
   plotxy,plot,x=x,y=y1,/time

   plotend,plot
END
```

Beispiel 5.13: PLOT Umgebung: plotxy: Markieren von Bildinformationen

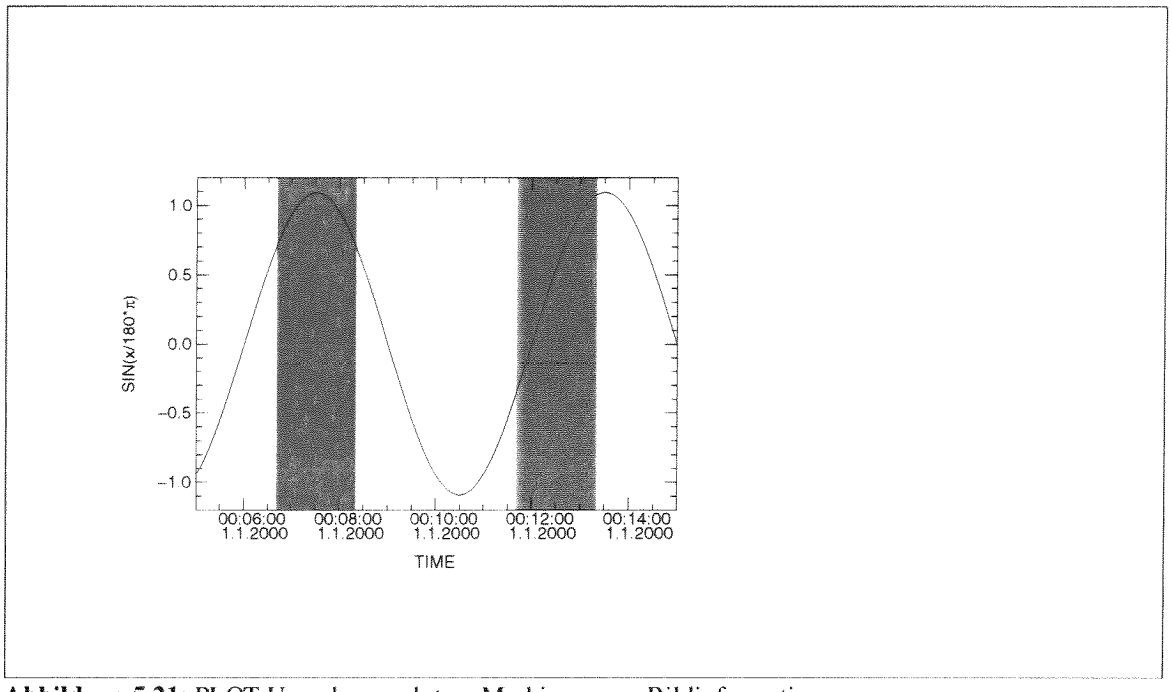


Abbildung 5.21: PLOT Umgebung: plotxy: Markieren von Bildinformationen

5.5.11 plot2d: 2-D Darstellung mit Hilfe eines Flächen-Contour-Plots

```
PRO example20
   RESTORE,get_profile_path()+'example20.sav'

   plotprepare,plot
   plot.stamp=0
   plot.psflag=1
   plot.psfile='example20'

   plotinit,plot
   plot2d,plot,x=x,y=y,z=zn[*,*,0]
   plotend,plot
END
```

Beispiel 5.14: PLOT Umgebung: plot2d

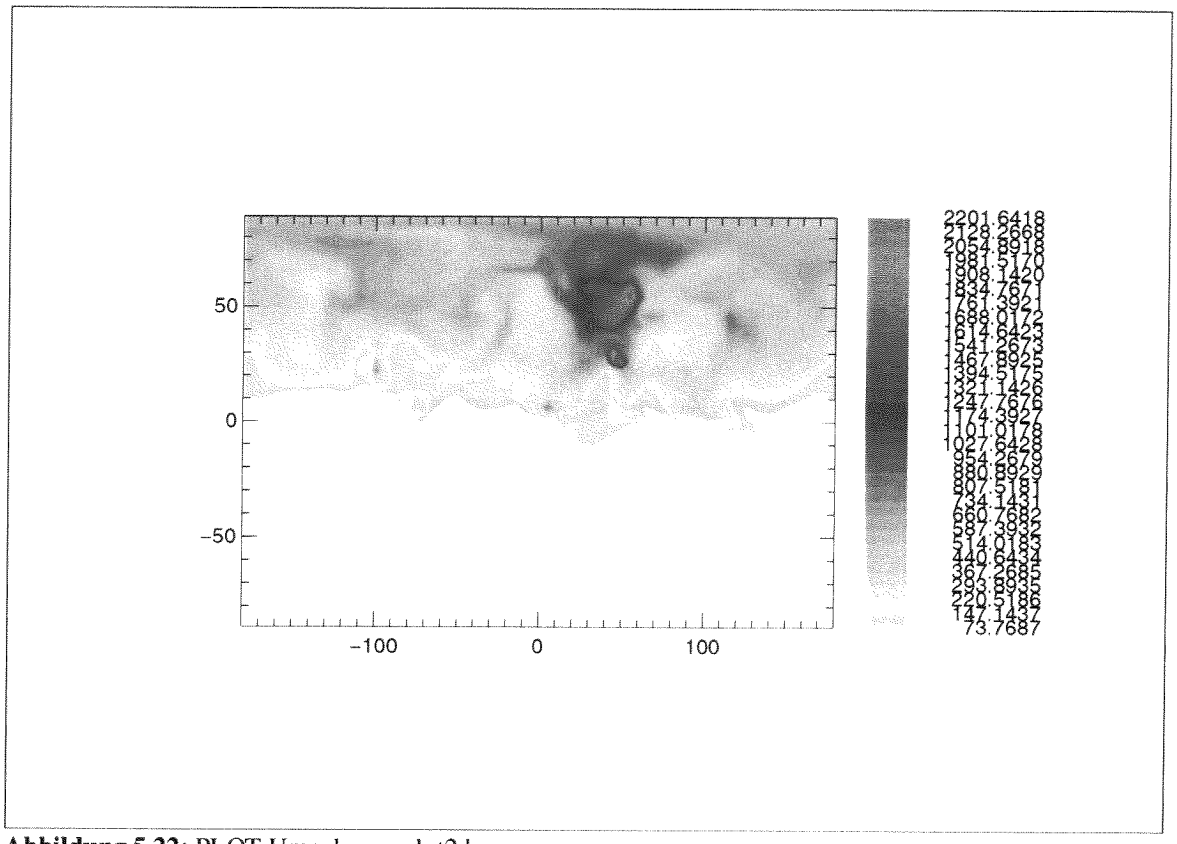


Abbildung 5.22: PLOT Umgebung: plot2d

5.5.12 plot2d: Verwendung von Level

```
PRO example21a
   RESTORE,get_profile_path()+'example20.sav'

   plotprepare,plot
   plot.stamp=0
   plot.psflag=1
   plot.psfile='example21a'
   plotinit,plot

   color_scheme,plot,scheme_code='ct_cyan_green_yellow_red'
   plot.nlevels=0
   defp2d,plot,'levels',[0,10,20,50,100,200,500,1000,2000]
   plot.cbar_title='C!D3!NH!D8!N [pptV]'

   plot2d  ,plot,x=x,y=y,z=zn[*,*,0]

   plotend,plot
END
```

Beispiel 5.15: PLOT Umgebung: plot2d: Verwendung von Level

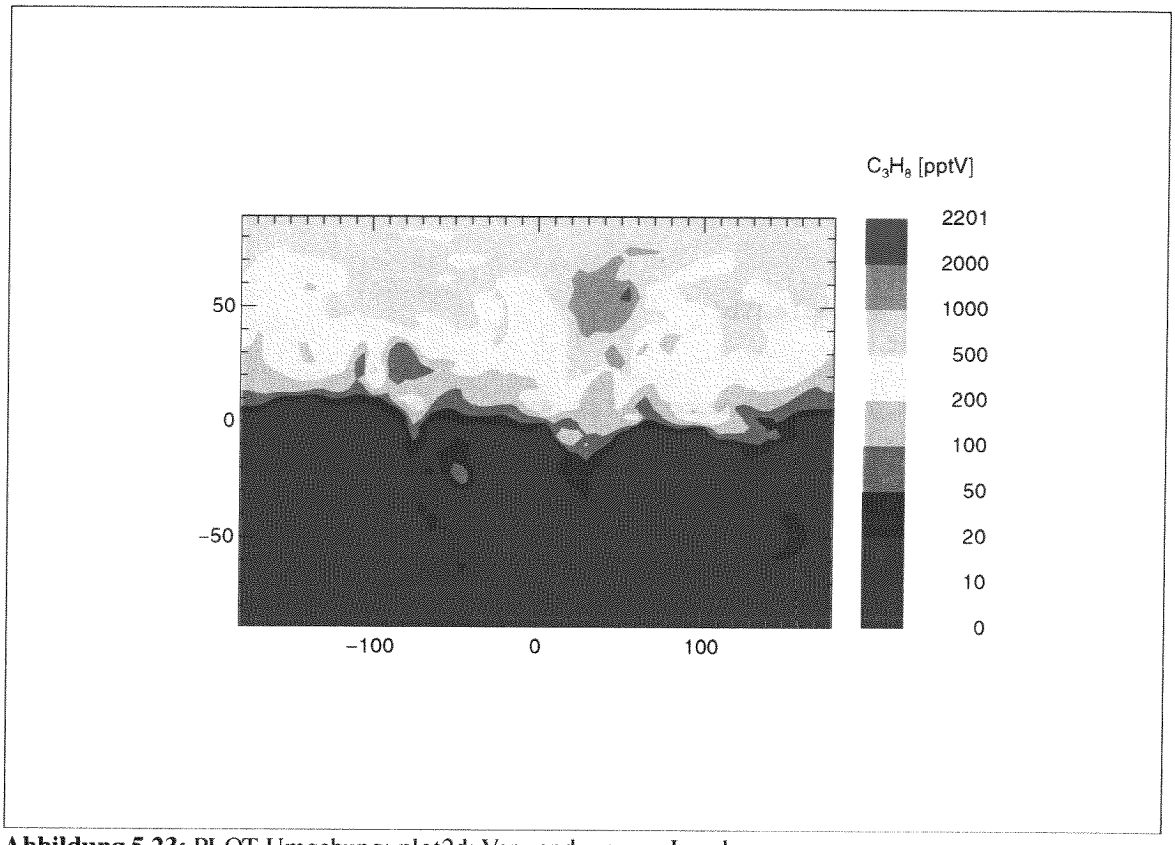


Abbildung 5.23: PLOT Umgebung: plot2d: Verwendung von Level

5.5.13 plotcell: Verwendung von Level

```
PRO example21b
   RESTORE,get_profile_path()+'example20.sav'

   plotprepare,plot
   plot.stamp=0
   plot.psflag=1
   plot.psfile='example21b'
   plotinit,plot

   color_scheme,plot,scheme_code='ct_cyan_green_yellow_red'
   plot.nlevels=0
   defp2d,plot,'levels',[0,10,20,50,100,200,500,1000,2000]
   plot.cbar_title='C!D3!NH!D8!N [pptV]'

   plotcell,plot,x=x,y=y,z=zn[*,*,0]

   plotend,plot
END
```

Beispiel 5.16: PLOT Umgebung: plotcell: Verwendung von Level

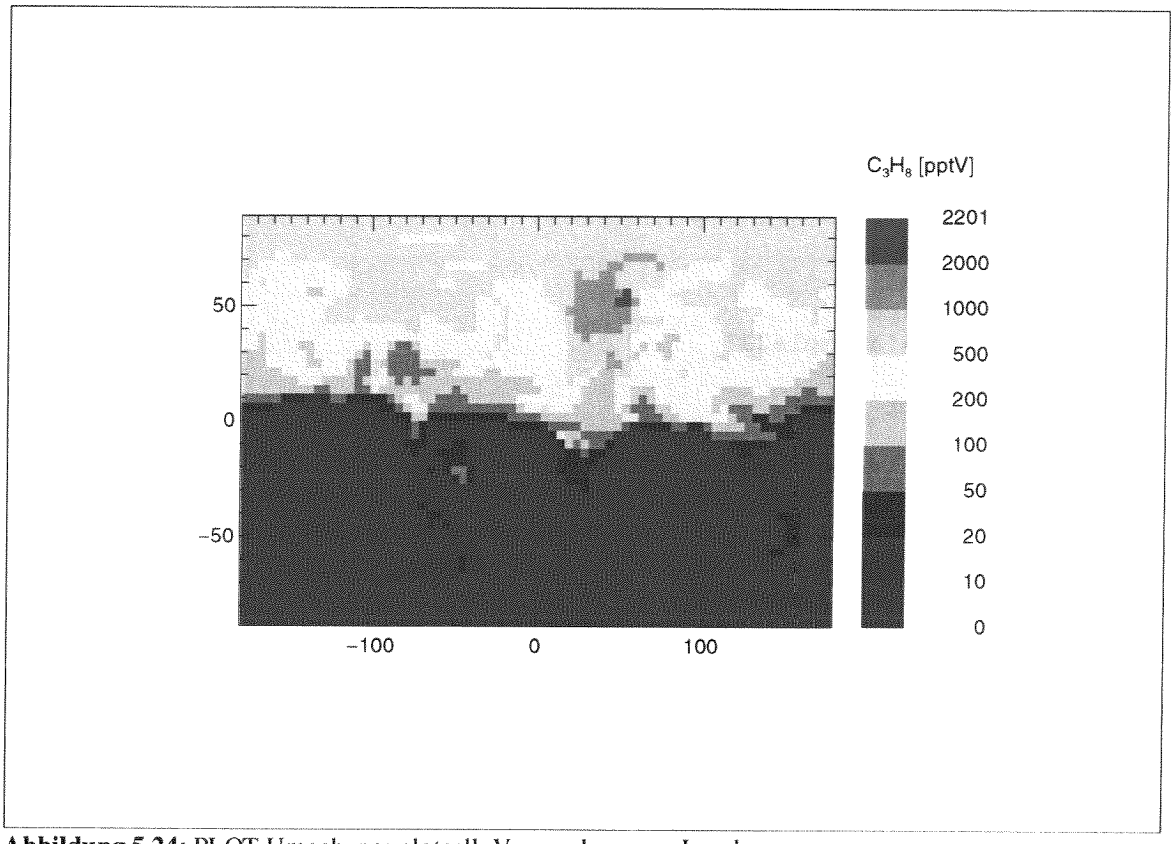


Abbildung 5.24: PLOT Umgebung: plotcell: Verwendung von Level

5.5.14 plotmap und plot2d

```
PRO example22
   RESTORE,get_profile_path()+'example20.sav'

   plotprepare,plot
   plot.stamp=0
   plot.psflag=1
   plot.landscape=2
   plot.psfile='example22'

   plotinit,plot

   plot.xtitle='Longitude'
   plot.ytitle='Latitude'

   plot.map_set.grid      =0
   plot.map_set.axis_label=1

   ;xp_map_set,plot ;========= automaticly replaced =========

   plotmap,plot
   plot.new=0

   color_scheme,plot,scheme_code='ct_cyan_green_yellow_red'
   plot.nlevels=0
   defp2d,plot,'levels',[0,10,20,50,100,200,500,1000,2000]
   plot.cbar_title='C!D3!NH!D8!N [pptV]'

   plot2d,plot,x=x,y=y,z=zn[*,*,0]
   plotend,plot
END
```

Beispiel 5.17: PLOT Umgebung: plotmap und plot2d

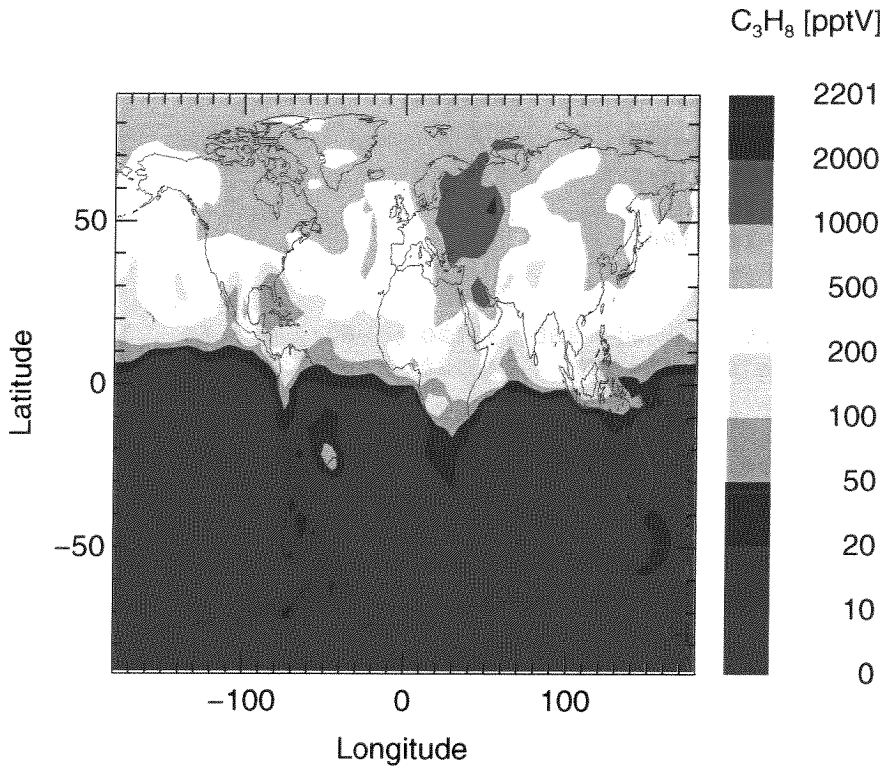


Abbildung 5.25: PLOT Umgebung: plotmap und plot2d

5.5.15 plotmap und plot2d: Umbenennung der Label

```
PRO example23
   RESTORE,get_profile_path()+'example20.sav'

   plotprepare,plot

   plot.stamp=0
   plot.psflag=1
   plot.landscape=2
   plot.psfile='example23'

   plotinit,plot

   plot.xtitle='Longitude'
   plot.ytitle='Latitude'
   plot.yrange=[-90,90]
   plot.xrange=[-180,180]
   plot.xticks=6
   plot.xminor=6
   plot.xtickv[0,0:6]   =[ -180 , -120 , -60 , 0 , 60 , 120 , 180 ]
   plot.xtickname[0,0:6]=['W'   ,'-120','-60','0','60','120','E'  ]
   plot.yticks=6
   plot.yminor=3
   plot.ytickv[0,0:6]   =[ -90 , -60 , -30 , 0 , 30 , 60 , 90 ]
   plot.ytickname[0,0:6]=['S'  ,'-60','-30','0','30','60','N' ]

   plot.map_set.grid      =0
   plot.map_set.axis_label=1
   plotmap,plot
   plot.new=0

   color_scheme,plot,scheme_code='ct_cyan_green_yellow_red'
   plot.nlevels=0
   defp2d,plot,'levels',[0,10,20,50,100,200,500,1000,2000]
   plot.cbar_title='C!D3!NH!D8!N [pptV]'

   plot2d,plot,x=x,y=y,z=zn[*,*,0]
   plotend,plot
END
```

Beispiel 5.18: PLOT Umgebung: plotmap und plot2d: Umbenennung der Label

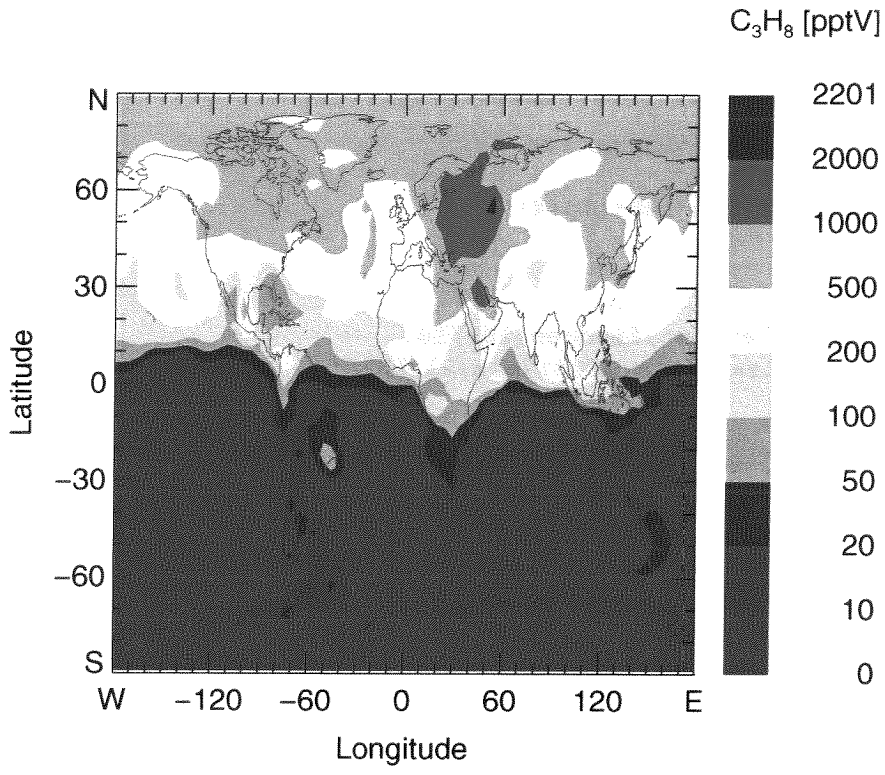


Abbildung 5.26: PLOT Umgebung: plotmap und plot2d: Umbenennung der Label

5.5.16 plotxy_2d

```
PRO example40
  RESTORE,get_profile_path()+'example1.sav'

  plotprepare,plot
  plot.psflag=1
  plot.psfile='example40'
  plot.stamp=0
  plot.landscape=0
  plotinit,plot

  plot.psym  =0
  plot.yrange=[-1.2,1.2]
  plot.color =plot.color_nc.red
  plot.xtitle='TIME'
  plot.ytitle='SIN(x/180*!7p!X)'
  plot.psym=208

  plot.nlevels=0
  defp2d,plot,'levels',findgen(10)/10.
  plot.cbar_title='SIN(x/180*!7p!X+2)!U2!N'


  plotxy_2d,plot,x=x,y=y1,z=y2,/time

  plotend,plot
END
```

Beispiel 5.19: PLOT Umgebung: plotxy_2d

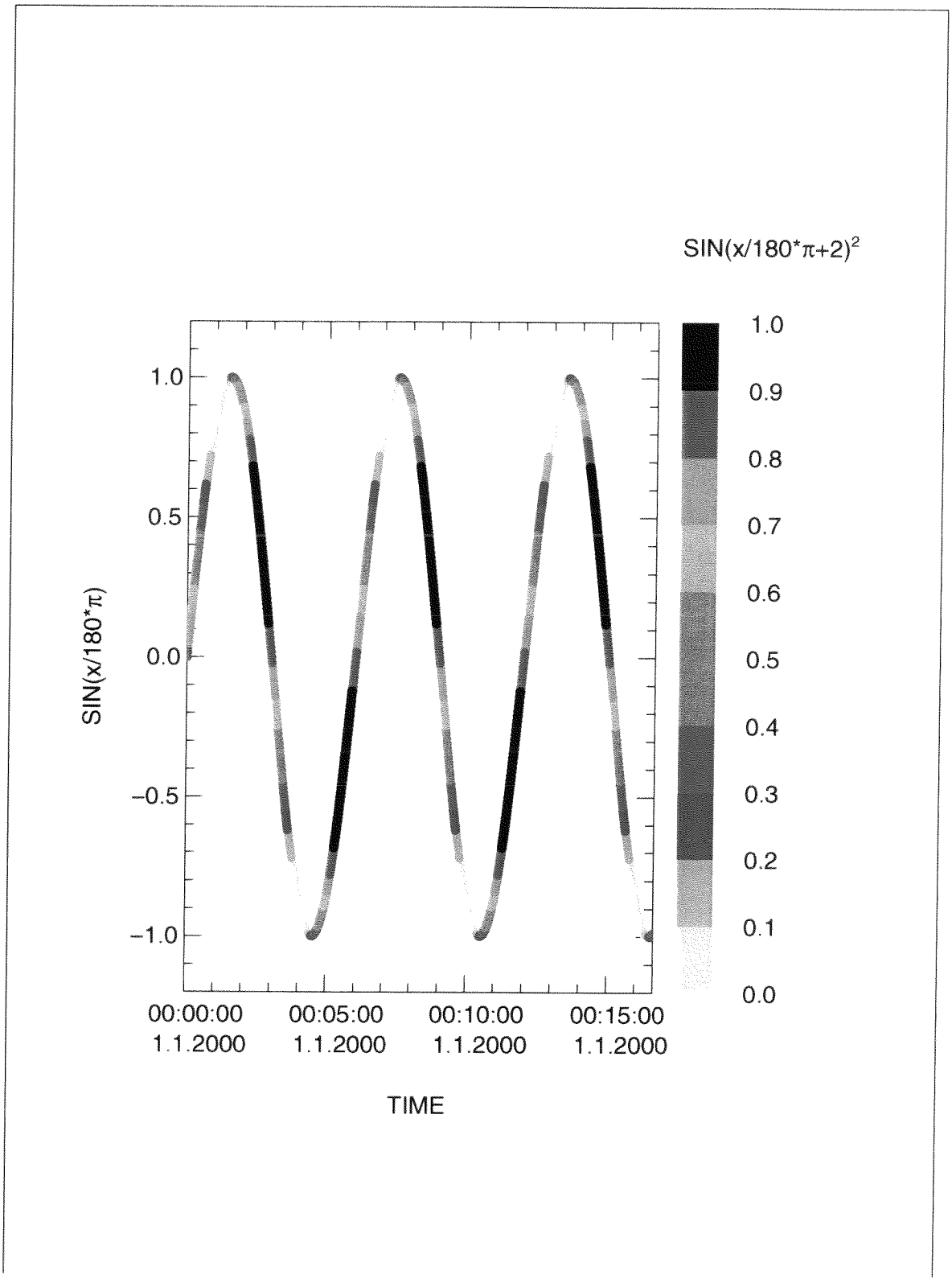


Abbildung 5.27: PLOT Umgebung: plotxy_2d

5.5.17 plot3d

```
PRO example50

   plotprepare,plot
   plot.psflag=1
   plot.psfile='example50'
   plot.logo.type=2
   plotinit,plot

   color_scheme,plot,scheme_code='ct_yellow_red_blue'
   plot.nlevels=0
   defp2d,plot,'levels',[-6,-4,-2,0,2]

   plot.irregular=0


   plot.fill=1
   x = INDGEN(10)
   y = INDGEN(10)
   z = DIST(10,10)
   shades=1-DIST(10,10)

   plot3d,plot,z=z,x=x,y=y,shades=shades

   plotend,plot

END
```

Beispiel 5.20: PLOT Umgebung: plot3d

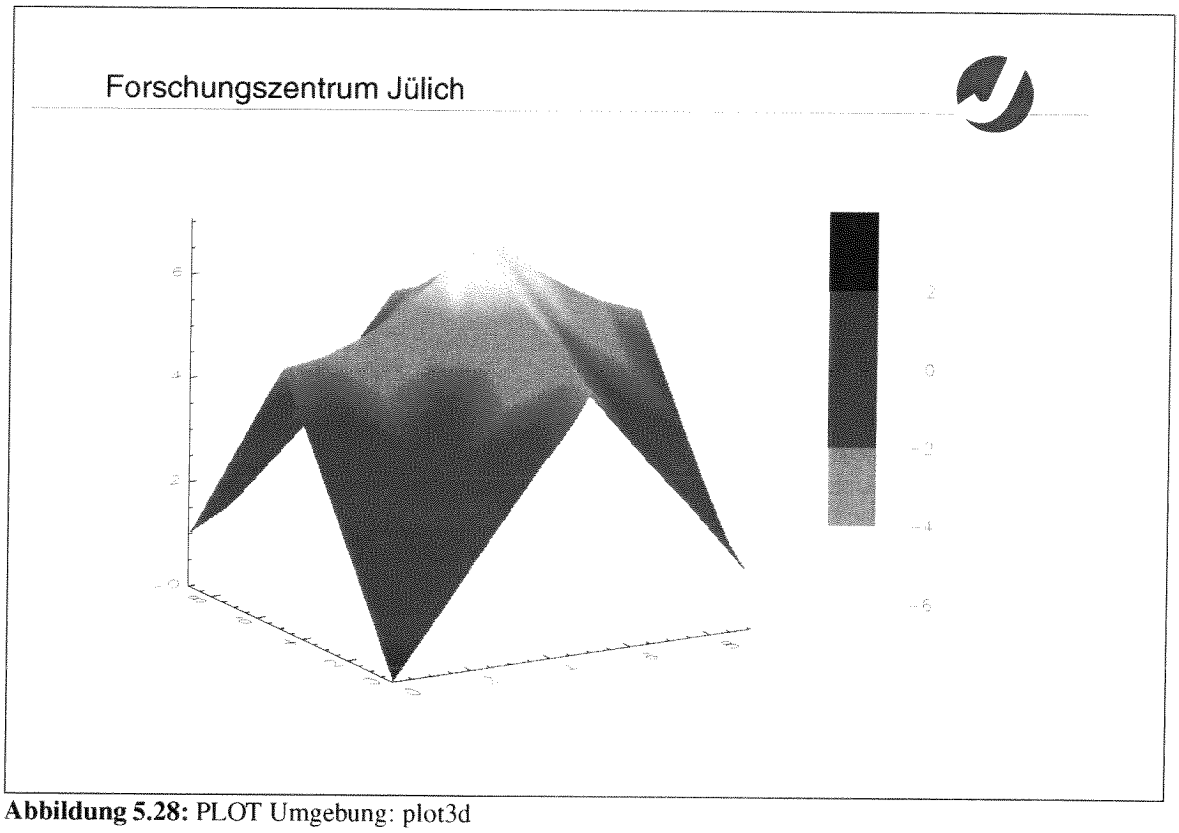


Abbildung 5.28: PLOT Umgebung: plot3d

5.5.18 Verschiedene plots auf einer Seite

Unterschiedliche Farbsysteme

Dieses Beispiel zeigt die Anordnung der verschiedenen Module auf einer Seite. Die `plot3d` und die `plot2d` Grafik verwenden unterschiedliche Farbsysteme.

```
PRO example60
   plotprepare,plot
   plot.psflag=1
   plot.psfile='example60'
   plot.rows=2
   plot.columns=2
   plotinit,plot
   color_scheme,plot,scheme_code=['ct_yellow_red_blue','ct_grey_191_32']
   plot.active_color_scheme=0
   plot.nlevels=0
   defp2d,plot,'levels',[-6,-4,-2,0,2]
   plot.irregular=0
   plot.plot_title.color=plot.color_nc.red
   plot.fill=1
   x = INDGEN(10)
   y = INDGEN(10)
   z = DIST(10,10)
   shades=1-DIST(10,10)
   plot.plot_title.name='PLOT3D'
   plot.plot_title.alignment=0.0
   plot3d,plot,z=z,x=x,y=y,shades=shades
   plot_text,plot

   plot.new=1
   plot.active_color_scheme=1
   defp2d,plot,'levels',[0,2,4,6,8]
   plot.plot_title.name='PLOT2D'
   plot2d,plot,z=z,x=x,y=y
   plot_text,plot

   plot.new=1
   plot.plot_title.name='PLOTXY'
   plot.color=plot.color_nc.green
   plot.symsize=2
   plot.psym=208
   plot.error_bar.s_hat=1
   plot.error_bar.s_color=plot.color_nc.black
   plotxy,plot,x=x,y=y,sx=0.5,sy=0.2
   plot_text,plot

   plot.new=1
   plot.plot_title.name='PLOTMAP'
   plotmap,plot
   plot_text,plot

   plotend,plot
END
```

Beispiel 5.21: PLOT Umgebung: plot3d, plot2d, plotxy, plotmap

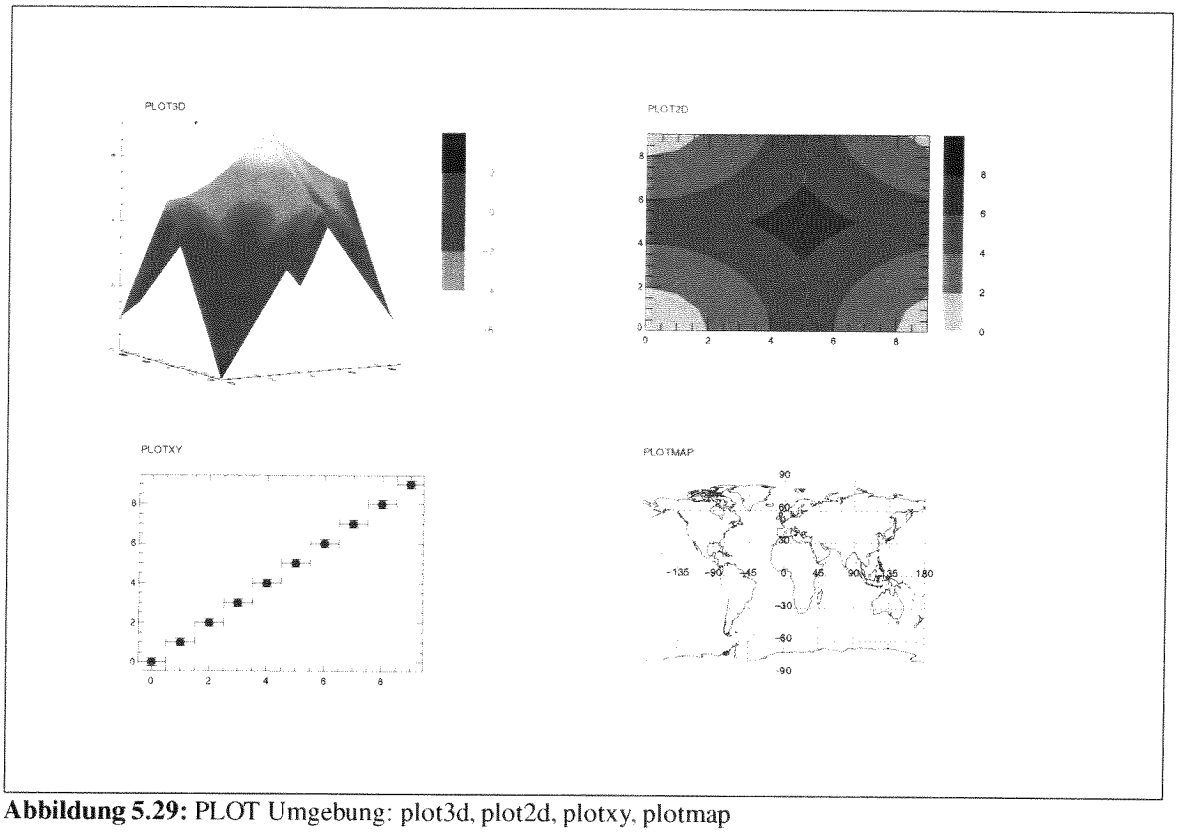


Abbildung 5.29: PLOT Umgebung: plot3d, plot2d, plotxy, plotmap

plot_map, plot2d und plotxy

Dieses Beispiel zegt auf einer Kartenprojektion eine Flugroute und eine gemessene Größe.

```
PRO example61
   RESTORE,get_profile_path()+'example20.sav'
   RESTORE,get_profile_path()+'track_stratoz3.sav'

   plotprepare,plot

   plotinit,plot

   plot.xtitle='geographische Länge'
   plot.ytitle='geographische Breite'
   plot.yrange=[-90,90]
   plot.xrange=[-180,180]
   plot.xticks=6
   plot.xminor=6
   plot.xtickv[0,0:6]   =[ -180 , -120 , -60 , 0 , 60 , 120 , 180 ]
   plot.xtickname[0,0:6]=['W'   ,'-120','-60','0','60','120','E'  ]
   plot.yticks=6
   plot.yminor=3
   plot.ytickv[0,0:6]   =[ -90 , -60 , -30 , 0 , 30 , 60 , 90 ]
   plot.ytickname[0,0:6]=['S'  ,'-60','-30','0','30','60','N' ]

   plot.map_set.grid      =0
   plot.map_set.axis_label=1
   plotmap,plot
   plot.new=0

   color_scheme,plot,scheme_code='ct_cyan_green_yellow_red'
   plot.nlevels=0
   defp2d,plot,'levels',[0,10,20,50,100,200,500,1000,2000]
   plot.cbar_title='C!D3!NH!D8!N [pptV]'

   plot2d,plot,x=x,y=y,z=zn[*,*,0]

   plot.new  =0
   plot.psym =0
   plot.thick=3
   plot.color=plot.color_nc.red
   plotxy,plot,x=lon,y=lat
   xyouts_box,0,87,'Flugroute: STRATOZ III',/data,alignment=0.5, $
    charsize=plot.charsize*1.5,boxcolor=plot.color_nc.black,$
    color=plot.color_nc.red,addspace=0.5

   plotend,plot
END
```

Beispiel 5.22: PLOT Umgebung: Kombination plot2d mit plotxy

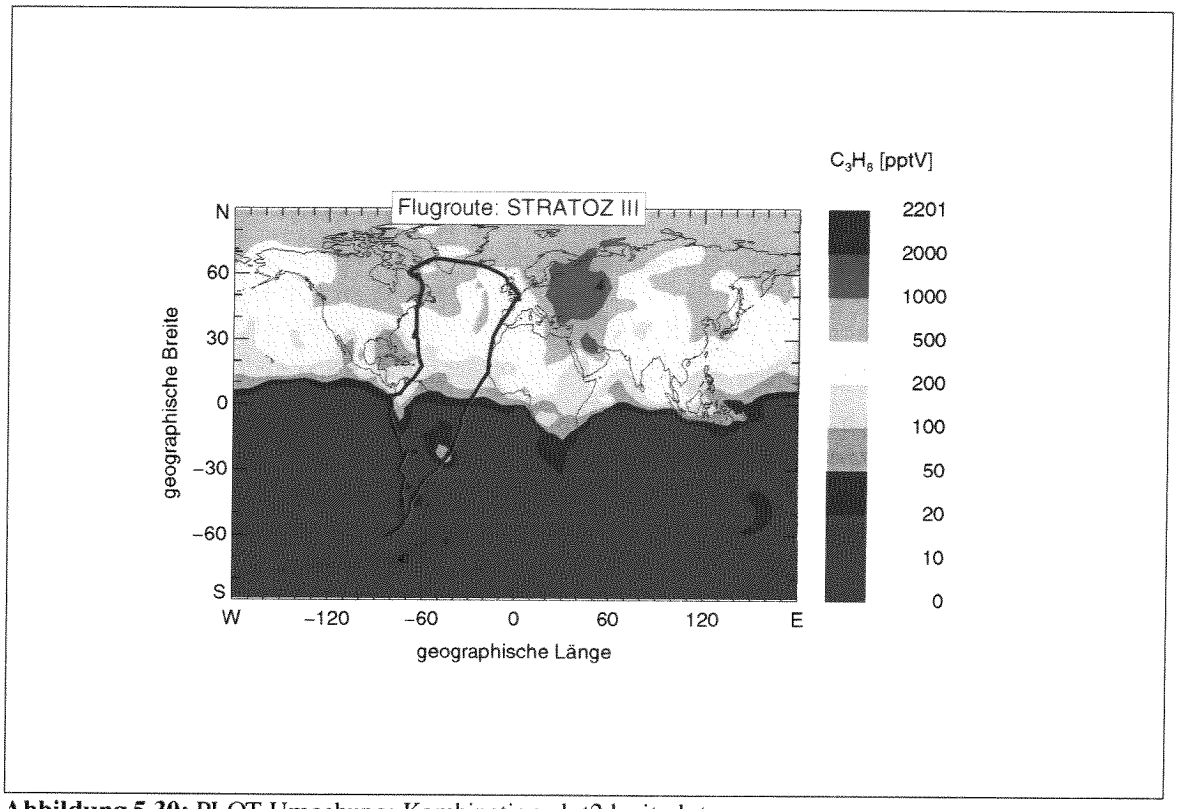


Abbildung 5.30: PLOT Umgebung: Kombination plot2d mit plotxy

5.5.19 Gif Film erzeugen

Ein Gif Film ermöglicht es im 'Fluss' zeitliche Änderungen komplexer Datenfelder zu veranschaulichen. Das zum Abspielen verwendete widget `x_animate` ist in der Lage, zeitsynchron verschiedene Filme wiederzugeben.

```
PRO example22,i
   RESTORE,get_profile_path()+'example20.sav'

   plotprepare,plot
   plot.stamp=0
   plot.psflag=12
   plot.landscape=2
   plot.psfile='example22'

   plotinit,plot

   plot.xtitle='Longitude'
   plot.ytitle='Latitude'

   plot.map_set.grid=0
   plot.map_set.axis_label=1

   plotmap,plot
   plot.new=0
   plot.title=n2s(i)
   plot.cbar_show_maxvalue=0
   plot.cbar_show_minvalue=0

   color_scheme,plot,scheme_code='ct_cyan_green_yellow_red'
   plot.nlevels=0
   defp2d,plot,'levels',[0,10,20,50,100,200,500,1000,2000]
   plot.cbar_title='C!D3!NH!D8!N [pptV]'

   plot2d,plot,x=x,y=y,z=zn[*,*,i]
   plotend,plot
END
PRO example70
   FOR i=0,10 DO BEGIN
      example22,i
      make_gif,'film',/multiple
   ENDFOR
   make_gif,'film',/CLOSE
   x_animate,file='film.gif'
END
```

Beispiel 5.23: PLOT Umgebung: GIF Film

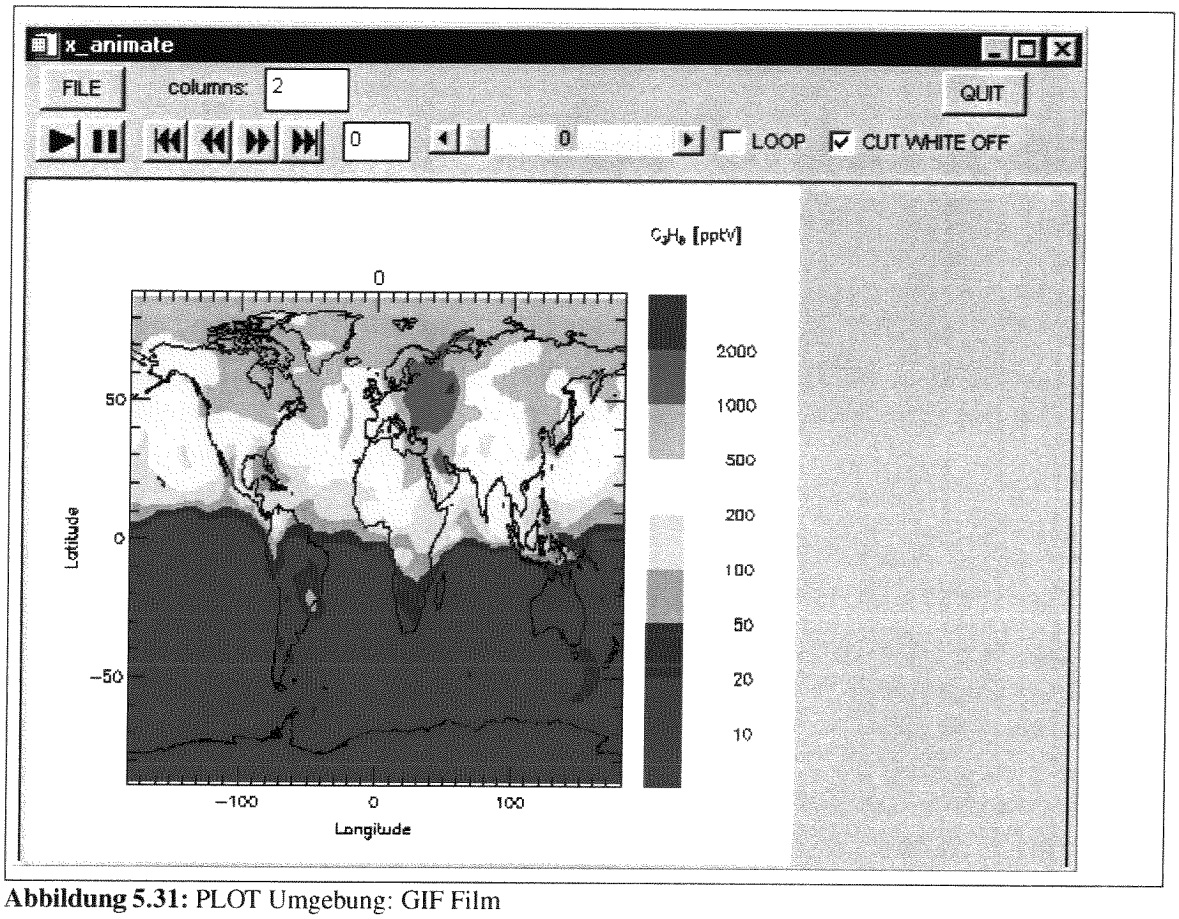


Abbildung 5.31: PLOT Umgebung: GIF Film

Kapitel 6

Die ICG1-Kampagnen-Daten

Dieser Abschnitt beschreibt die Organisation der ICG-1 Kampagnen-Daten.

Die ICG1-Kampagnen-Daten sind Daten von Flugzeug- oder Ballon-Messkampagnen. Diese Daten werden in einer Baumstruktur gespeichert und auch nur dort bearbeitet. Das oberste Verzeichnis ist das Kampagnenverzeichnis z.B.: stream1997.

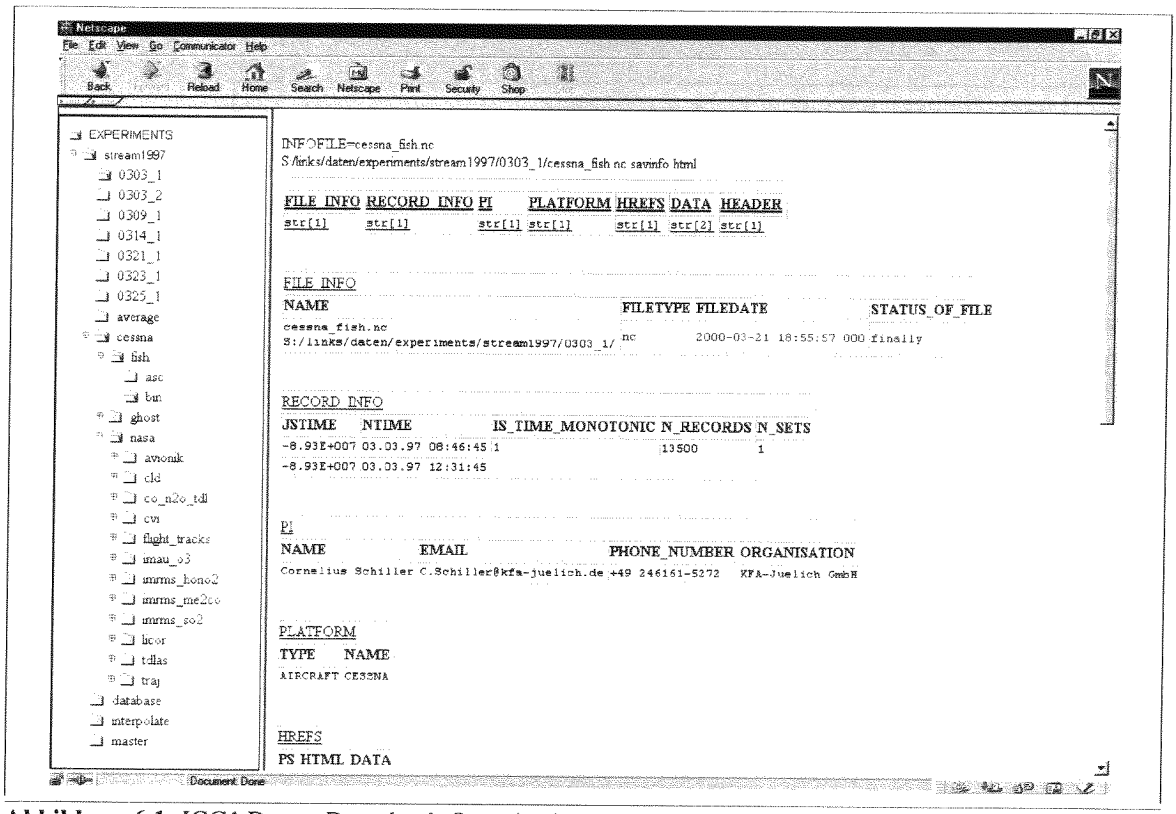


Abbildung 6.1: ICG1 Daten: Datenbank Organisation

Unterhalb dieses Kampagnen-Verzeichnisses werden eine Reihe von Unterverzeichnissen angelegt.

1. Plattformverzeichnisse, z.B.: cessna
2. Missionsverzeichnisse, z.B.: 0303_1
3. Katalogisierungsverzeichnisse, z.B.: database
4. Resultatverzeichnisse, z.B.: master

In den Plattformverzeichnissen (z.B. cessna) sind Unterverzeichnisse unserer Experimente und Unterverzeichnisse der Austauschdatenformate (z.B. nasa, fish). Ausgewertete Datendateien sind in die Missionsverzeichnisse symbolisch mit dem Plattform- und dem Experimentnamen gelinkt. Da-

durch ist ersichtlich, welche Datei welche Daten beinhaltet. Aus diesen Dateien wird auch eine Zusammenfassung der Daten in Abhängigkeit zur Zeit (MASTER) erstellt.

Durch eine Vereinheitlichung der Namen der einzelnen gemessenen Substanzen können Auswertungen automatisch erfolgen. Gleichzeitig entsteht dadurch eine Datenbank.

Für die Bezeichnungen der einzelnen gemessen Substanzen verwenden wir die vom ASAD-Modell siehe [*G. D. Carver et al.*(1997)] vorgegebene Namenskonvention.

Verschiedene Beispiele gemäß der ASAD-Namenskonvention:

-ONO	für NO2
-ONON2	für NO3
-O2NO2	für NO4
-OO	für Peroxide
Me	für Methyl CH3-
Et	für Ethyl CH3CH2-
Pr	für Propyl CH3H7- (i-iso, n-normal)
Bu	für Butyl CH4H9- (i,n,t)
-CHO	für Aldehyde

Die Halogenkohlenwasserstoffe werden entsprechend der Standardnamenskonvention für Fluor-Chlor-Kohlenwasserstoffe wie folgt wiedergegeben:

F	Freon (Halogen mit einem Kohlenstoffatom)
H	andere Halone
R	Halogenkohlenwasserstoff Radikal, Kohlenwasserstoffatom aktiviert
erste Stelle:	Anzahl der C Atome minus 1
zweite:	Anzahl der H Atome plus 1
dritte:	Anzahl der F Atome
vierte:	Cl Atome
fünfte:	Br Atome
t:	bei der End-Gruppe: -CF3
d:	bei der End-Gruppe: -CF2A
m:	bei der End-Gruppe: -CFAB

Br	Br2	BrCl	BrO	BrONO
BrONO2	BrSH	C	C2Cl4	C2H2
C2H2Cl	C2H2OH	C2H4	C2H4Cl	C2H6
C2HCl3	C2HO	C3H6	C3H6OH	C3H8
C4F8	CCl3	CCl3CHO	CCl3CO	CCl3O
CCl3O2NO2	CCl3OO	CCl4	CF2Br	CF2Br2
CF2Cl	CF2Cl2	CF2ClBr	CF2ClO	CF2ClO2NO2
CF2ClOO	CF3	CF3Br	CF3CHO	CF3CO
CF3COCl	CF3COF	CF3Cl	CF3O	CF3O2NO2
CF3OO	CF4	CFCl2	CFCl2O	CFCl2O2NO2
CFCl2OO	CFCl3	CH2	CH2Br	CH2COCl
CH2Cl	CH2Cl2	CH2ClCHO	CH2F	CH2F2
CH2FCl	CH2I	CH2OH	CH2OOH	CH3
CH3Br	CH3Cl	CH3F	CH3I	CH4
CHBr3	CHCl2	CHCl2CHO	CHCl3	CHF2
CHF2Br	CHF2Cl	CHF3	CHFCl	CHFCl2
CN	CO	CO2	COCl2	COF2
COFCl	CS	CS2	Cl	Cl2
Cl2O	Cl2O2	Cl2O3	ClCO	ClNO
ClNO2	ClO	ClO3	ClONO	ClONO2
ClOO	ClSH	Et	EtCHO	EtO
EtO2NO2	EtOH	EtONO	EtONO2	EtOO
EtOOEt	EtOOH	F	F2	FNO
FO	FO2	FONO	FONO2	FSH
H	H1133d	H1133t	H11402d	H1142d
H1142t	H1151t	H116	H12311t	H1231mOF22
H1231tOF22	H1232t	H12401t	H1241t	H125
H1322d	H1331t	H134d	H134t	H1403t
H1412m	H1421d	H143d	H143t	H152d
H152m	H161	H2	H2252d	H2252t
H226iOF31	H2432m	H2O	H2O2	H2S
H2SO4	HBr	HCHO	HCN	HCO
HCO2H	HCOCHO	HCOCl	HCOF	HCl
HF	HI	HNO	HO2	HO2NO2
HOBr	HOCH2CH2	HOCH2CH2O	HOCH2CH2OH	HOCH2CH2OO
HOCH2CHO	HOCH2CO	HOCH2O	HOCH2OH	HOCH2OO
HOCH2OOH	HOCHCHO	HOCS2	HOCl	HOI
HONO	HONO2	HOSO2	HS	HSNO
HSO	HSO2	HSOH	I	I2
INO	IO	IONO	IONO2	Me2CO
Me2S	MeCHO	MeCHOH	MeCN	MeCO
MeCO2	MeCO2H	MeCO3	MeCO3H	MeCOCH2
MeCOCH2O	MeCOCH2OH	MeCOCH2OO	MeCOCHO	MeCOCO
MeCOCl	MeO	MeO2NO2	MeOH	MeONO
MeONO2	MeOO	MeOOH	MeOOMe	MeS
MeSCH2	MeSH	MeSNO	MeSO	MeSO2
MeSS	MeSSMe	MeSSO	N	N2
N2O	N2O5	NCO	NH	NH2
NH3	NO	NO2	NO2SO3	NO3
Na	NaCO3	NaCl	NaHCO3	NaO
NaO2	NaO3	NaOH	O(3P)	O(1D)
O2	O3	OCS	OClO	OH
PAN	R11311t	R1132t	R11401t	R1141t

Tabelle 6.1: ICG1 Daten: Beispiele der ASAD Namenskonvention

Meteorologische Größen werden wie folgt bezeichnet:

meterologische Größe:	**Bezeichner:**
angle of attack	AOA
angle of sideslip	AOS
GPS altitude	GPS:ALT
groundspeed	GS
height	HGT
horizontal wind direction	HWD
horizontal wind speed	HWS
nickangle	NA
pressure altitude	P_ALT
cabin press	P_CABI
static pressure	PRESS
roll angle	RA
true air speed	TAS
static air temperature	TEMP
heading	THDG
potential temperature	THETA
latitude	LAT
longitude	LON
vertical wind component	VWC
wind angle	WA
windspeed	WS

Tabelle 6.2: ICG1 Daten: Bezeichnung meteorologischer Größen

6.1 Administration der Kampagnen-Daten

Im ICG-1 werden die Informationen der Kampagnen-Daten in einer relationalen Datenbank gespeichert. Dadurch werden folgende Zielsetzungen erreicht:

- alle Benutzer verwenden die gleichen Daten Dateien;
- Operationen mit den Daten sind nachvollziehbar;
- an nur einer Stelle ist eine Aktualisierung oder Änderung der Daten nötig;
- die Datensicherung wird vereinfacht.

Für die Organisation bzw. Administration der Kampagnen-Daten werden folgende Programme verwendet:

6.1.1 icg1_do_link2mission

Mit dieser Routine werden unter UNIX die symbolisch gelinkten Dateien in den Missionsverzeichnissen erstellt.

```
icg1_do_link2mission,'STREAM1997','cessna/fish/bin',$
  filter='ff*.dat.CALC.bin.nc',/use
```

Beispiel 6.1: ICG1 Daten: icg1_do_link2mission

6.1.2 icg1_database

Diese Routine erstellt eine Datenbank von den Daten-Dateien einer Kampagne. Diese Datenbank wird dann von `icg1_data` (siehe Kapitel 6.2.2 auf Seite 132) verwendet um Dateien auszuwählen. Zusätzlich werden die einzelnen Datensätze in HTML-Dateien wiedergegeben.

```
icg1_database,'STREAM1997'
```

Beispiel 6.2: ICG1 Daten: icg1_database

6.1.3 icg1_write_master_file

Aufgrund der Katalogisierung der Dateien in den Missionsverzeichnissen werden die MASTER-Dateien erstellt. Diese Dateien enthalten pro Mission alle gemessenen Parameter in einer Datei. Zur Aufnahme der MASTER-Dateien in die Datenbank, muss sie derzeit mit `icg1_database` neu aufgebaut werden.

```
icg1_write_master_file,'STREAM1997'
```

Beispiel 6.3: ICG1 Daten: icg1_database

6.1.4 icg1_database_make_plots

Durch diese Routine werden Übersichtsgraphiken der Kampagnen-Daten-Dateien erstellt. Dadurch erhält jeder einen Überblick über die vorhandenen Größen. Das Beispiel zeigt die Erstellung der Graphiken der MASTER-Dateien.

```
icg1_database_make_plots,'STREAM1997',/master_base
```

Beispiel 6.4: ICG1 Daten: icg1_database_make_plots

Die erste Seite einer solchen Übersicht zeigt Abbildung 6.2:

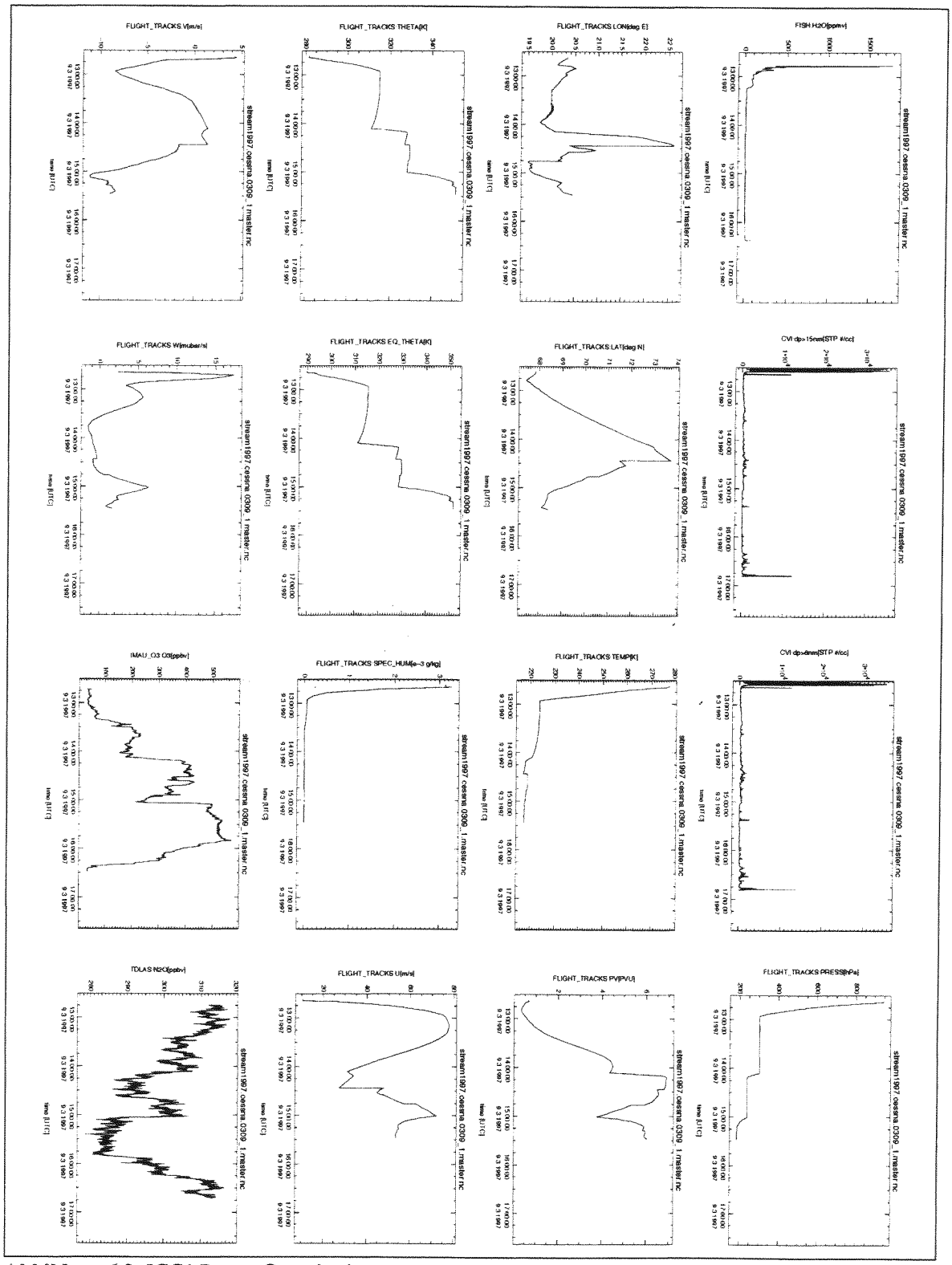


Abbildung 6.2: ICG1 Daten: Organisation

6.1.5 icg1_make_dirtree

Diese Routine erstellt eine HTML Datei. Diese enthält in Form einer Baum-Struktur in mehreren Fenstern den Inhalt eines Kampagnen-Verzeichnisses (siehe Abbildung 6.1 auf Seite 125).

6.2 Benutzung der Kampagnen Daten

Durch die Organisation der Kampagnen-Daten in einer relationalen Datenbank, sind Auswertungen im batch-Prozess möglich. Die gewünschten Datendateien werden mit der Funktion `icg1_data` (siehe Kapitel 6.2.2 auf der nächsten Seite) lokalisiert. So können Auswerteroutinen, die für die STREAM1997 Kampagne geschrieben wurden, mit den Daten der STREAM1998 Kampagne verwendet werden. Weiterhin können durch Abfrage der Datenbank, Randbedingungen für eine bestimmte Auswertung ermittelt werden. So ist es z.B. möglich Anfangs- und End-Zeiten der ausgewählten Daten für alle Parameter aus der Datenbasis zu holen. Zum Visualisieren der Eintragungen der Datenbank dient die graphische Benutzer-Schnittstelle `icg1_handle_db` (siehe Kapitel 6.2.1).

6.2.1 icg1_handle_db

Diese Routine erlaubt es, mit Hilfe eines widgets verschiedene Sichten auf eine Datenbasis zu erstellen. Es lassen sich auch HTML-Seiten von gefilterten Eintragungen erstellen. Die erstellten Filter lassen sich als IDL-Quelltext abspeichern und auch wieder laden.

```
icg1_handle_db,'stream1997'
```

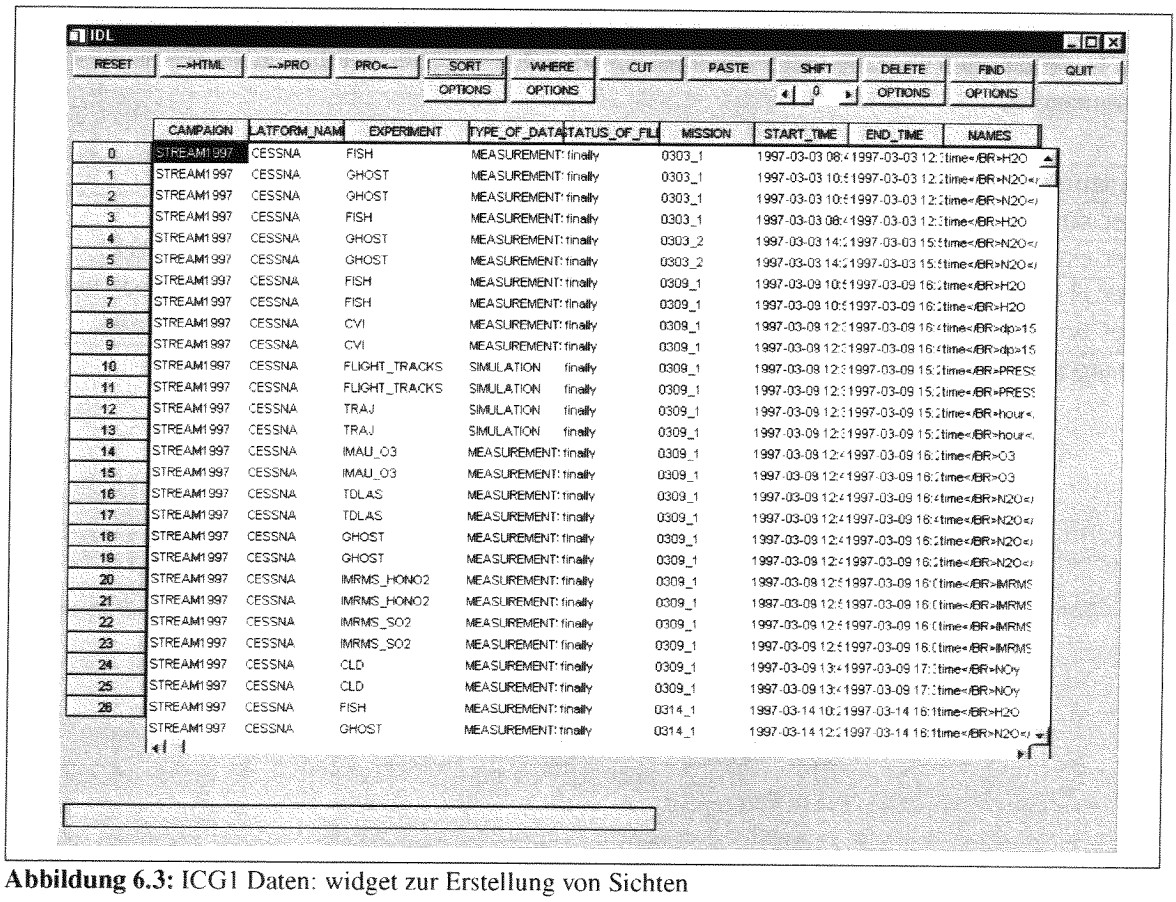


Abbildung 6.3: ICG1 Daten: widget zur Erstellung von Sichten

6.2.2 icg1_data()

Mit dieser Funktion kann man Dateien aus der Datenbasis nach verschiedenen kombinierbaren Kriterien auswählen. Diese Funktion gibt die den Kriterien entsprechenden Dateinamen zurück. Die voreingestellte Suche erfolgt nur in den Datendateien der Missionsverzeichnisse. Die Dateien können dann mit den Lesefunktionen z.B.: `readncdf`, `ncdf_files2icg_struct` ausgelesen werden. Folgende Kriterien sind möglich:

Steuerwort:	**Bezeichnung:**
mission	die Missionskennung z.B: 0723_1
between_time	der verwendete Zeitbereich z.B: ['1997-03-11 00:00:00','1997-03-15 00:00:00']
type_of_data	der Typ der Daten
platform_name	der Name der verwendeten Plattform
experiment	der Name des Experiments
campaign	der Name der Kampagne an der das Experiment teilgenommen hat

Tabelle 6.3: ICG1 Daten: icg1_data Auswahlkriterien

Weiterhin kann man bestimmen wo überall gesucht wird:
Mit einem dieser Befehle wird die Voreinstellung ausser Kraft gesetzt.

Steuerwort:	**Bezeichnung:**
whole_base	alles wird durchsucht
master_base	nur in den MASTER Dateien wird gesucht

Tabelle 6.4: ICG1 Daten: icg1_data Auswahl der Datenbasis

Um Informationen über die in der Datenbasis vorhandenen Daten zu erhalten, helfen nachfolgende Kommandos.

Argument:	**Wert:**
get_missions	auf dieser Variable erhält man alle Missionen zurück
get_experiments	auf dieser Variable erhält man alle beteiligten Instrumente zurück.
get_platform_names	auf dieser Variable erhält man alle verwendeten Plattform Namen zurück.
get_time	auf dieser Variable erhält man alle Anfangs- und Endzeiten zurück.

Tabelle 6.5: ICG1 Daten: icg1_data Informationen abfragen

Experimente in der Datenbank

Das nachfolgende Beispiel zeigt, wie man Informationen über beteiligte Experimente erhält.

```
a=icg1_data(campaign='stream1997', get_experiments=experiments)

printtab,experiments
CLD
CVI
FISH
FLIGHT_TRACKS
GHOST
IMAU_O3
IMRMS_HONO2
IMRMS_ME2CO
IMRMS_SO2
LICOR
TDLAS
```

Beispiel 6.5: ICG1 Daten: Datenbank benutzen, get_experiments

Plattformen des Experiments

Das Beispiel 6.6 zeigt, wie man Informationen über die verwendeten Plattform-Namen erhält.

```
a=icg1_data(campaign='stream1997', experiment='FISH', $
  get_platform_names=platforms)

PRINT,platforms
CESSNA
```

Beispiel 6.6: ICG1 Daten: Plattformen des Experiments

Verwendung der Missionen

Mit diesem Beispiel wird gezeigt, wie man Daten-Dateien von einer Mission in einer ICG-Daten-Struktur abbildet.

```
a=icg1_data(campaign='stream1997', get_missions=missions)

PRINT,missions
0303_1 0303_2 0309_1 0314_1 0321_1 0323_1 0325_1

data=ncdf_files2icg_struct(icg1_data(campaign='stream1997',$
    mission=missions[0]))

struct_tree,data
```

Beispiel 6.7: ICG1 Daten: Datenbank benutzen,get_missions

Schematischer Aufbau der Struktur aus Beispiel 6.7.

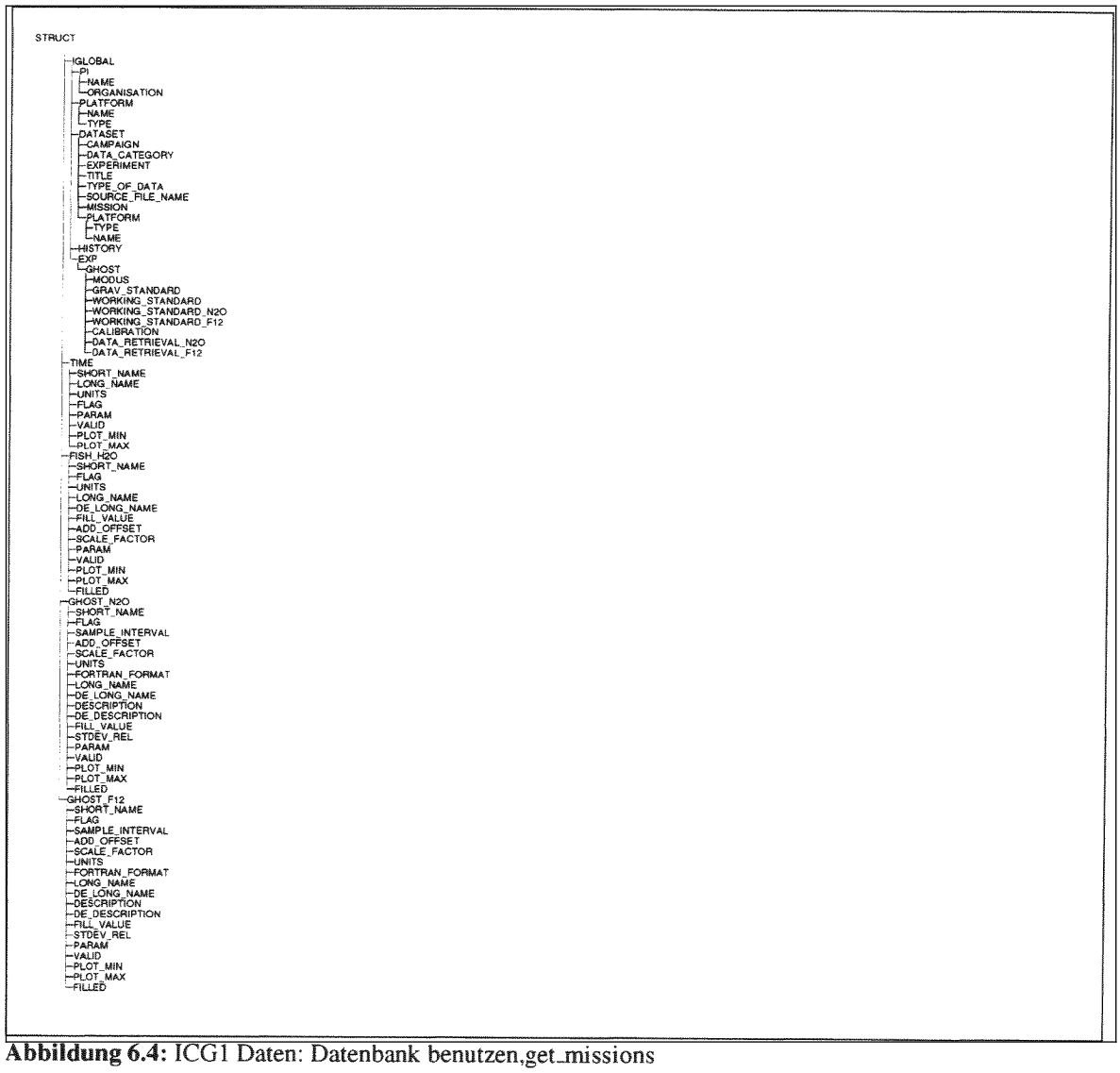


Abbildung 6.4: ICG1 Daten: Datenbank benutzen,get_missions

Verarbeitung eines Experiments

Dieses Beipiel zeigt, wie mehrere Daten-Dateien eines Experiments auf einer ICG-Daten-Struktur eingelesen werden.

```
files=icg1_data(campaign='stream1997', experiments='FISH')

data=ncdf_files2icg_struct(files, /concat_names)

struct_tree,data
```

Beispiel 6.8: ICG1 Daten: Datenbank benutzen, Zusammenfügen von Daten

Schematischer Aufbau der Struktur aus Beispiel 6.8.

```
STRUCT
    ├─!FILENAME
    ├─!GLOBAL
    │  ├─HISTORY
    │  ├─PI
    │  │  ├─NAME
    │  │  ├─WORKGROUP
    │  │  ├─ORGANISATION
    │  │  ├─INSTITUTE
    │  │  ├─ADDRESS
    │  │  ├─PHONE_NUMBER
    │  │  └─EMAIL
    │  ├─DATASET
    │  │  ├─TITLE
    │  │  ├─DATA_CATEGORY
    │  │  ├─DATA_ORIGIN
    │  │  ├─TYPE_OF_DATA
    │  │  ├─SOURCE_FILE_NAME
    │  │  ├─CREATION_TIME
    │  │  ├─MODIFICATION_TIME
    │  │  ├─CAMPAIGN
    │  │  ├─MISSION
    │  │  ├─STATUS_OF_FILE
    │  │  ├─EXPERIMENT
    │  │  ├─START_OF_TIME
    │  │  └─END_OF_TIME
    │  └─PLATFORM
    │     ├─TYPE
    │     └─NAME
    ├─TIME
    │  ├─SHORT_NAME
    │  ├─FLAG
    │  ├─UNITS
    │  ├─LONG_NAME
    │  ├─DE_LONG_NAME
    │  ├─ADD_OFFSET
    │  ├─SCALE_FACTOR
    │  └─PARAM
    └─H2O
       ├─SHORT_NAME
       ├─FLAG
       ├─UNITS
       ├─LONG_NAME
       ├─DE_LONG_NAME
       ├─FILL_VALUE
       ├─ADD_OFFSET
       ├─SCALE_FACTOR
       ├─PARAM
       ├─VALID
       ├─PLOT_MIN
       ├─PLOT_MAX
       └─FILLED
```

Abbildung 6.5: ICG1 Daten: Datenbank benutzen, Zusammenfügen von Daten

Kapitel 7

Zusammenfassung

In diesem Band werden die prinzipiellen Eigenschaften und Möglichkeiten der ICG-IDL-Bibliothek vorgestellt. Für die Bearbeitung von Daten wird die Definition und die Verwendung der ICG-Daten-Struktur erklärt. Diese beinhaltet Vereinbarungen, die für das Erstellen einer relationalen Datenbank für Kampagnendaten verwendet werden. Um die Daten zu Visualisieren werden eine HTML-Umgebung und eine PLOT-Umgebung vorgestellt. Die PLOT-Umgebung wird verwendet um standardmäßig hochwertige Grafiken zu erzeugen. Hier werden die Ausgabemöglichkeiten: Bildschirmausgabe, Postscript und GIF-Film gezeigt. Die HTML-Umgebung dient der Veröffentlichung der Daten und Bilder im WWW. Mit Hilfe der Datenstruktur-, der Datenbank- und den Visualisierungs-Modulen sind Auswertungen im batch-Prozess möglich.

Index

Forschungszentrum Jülich

Jül-3786
Juli 2000
ISSN 0944-2952