jbldarf 徹底ガイド

日本語 第1.2版

石崎 欣尚
東京大学大学院理学系研究科
ishisaki@miranda.phys.s.u-tokyo.ac.jp

平山 昌治
東京大学大学院理学系研究科
hirayama@astro.isas.ac.jp

石田 学
宇宙科学研究所宇宙圏研究系
ishida@astro.isas.ac.jp

1995年4月01日　第1.0版
1995年7月20日　第1.1版
1995年8月27日　第1.2版

目次

はじめに

jbldarfは簡単なパラメーターの設定により、SISとGISのARF(Auxiliary/ Ancillary Response Function[1])を作るソフトです。このマニュアルでは、**jbldarf**の使い方から適用限界にいたるまで、**jbldarf**に関するさまざまな解説がなされいます。内容は、現時点での最新バージョンである2.10に基づいて書かれています。jbldarf-1.00と比べて新しくなった節には、欄外に◁U▷(UPDATED)や◁N▷(NEW)のマークをつけてあります。

第1章では、**jbldarf**のインストール方法を説明します。あなたが使っている計算機に**jbldarf**をインストールするときにお読み下さい。すでに誰かの手によってインストールがされているならここを読む必要はありません。インストールされているかどうかわからない人は、あなたが使っている計算機の管理者に問い合わせてみて下さい。

第2章では、**jbldarf**の具体的な操作法や、リージョン、ポジションなどの基本的な概念について説明されています。初心者の方は、ここから読まれることをお勧めします。また、**jbldarf**に十分慣れている方でも、忘れてしまったコマンドの使い方を思い出すためのコマンドリファレンスとしてお役立て下さい。

第3章には、**jbldarf**をより良く理解していただくための詳しい説明があります。ARFの概念やファイル構造が記述され、**jbldarf**がどのようなデータに基づいてどのような計算をしてARFを作成しているのかを説明しています。その上で、現時点で考えうる問題点や**jbldarf**の適用限界についても触れ、やや高度な応用についても紹介します。この章では、第2章での知識を前提にしますので、始めて読まれる方は、この章を読まれる前に第2章を読んで下さい。また、**jbldarf**をお使いになられる方はこの章までは必ずお読みになって下さい。

第4章では、**jbldarf**のリリースキットに附属している、ARF関連のツールを紹介します。ツールの使い方を知りたい時など必要に応じてお読み下さい。

第5章では、これまでにユーザの皆様から著者宛に寄せられたさまざまなご意見の中から、比較的よく耳にする疑問や質問を集めて作った「質問 & 回答」集です。必要に応じてご参考になさって下さい。

jbldarfに関するご意見やご質問などは著者までお知らせ下さい。解析してみて始めてわかった、レスポンスに関する新事実などもお待ちしております。**jbldarf**にみなさんのご意見や解析結果などを反映させ、より正確で、より使いやすい**jbldarf**にしていきましょう。

---

[1] XSPECではauxiliary、FTOOLSではancillaryという単語が使われているようである。

第 1 章

インストール

この章では **jbldarf-2.10** のインストールの方法について説明します。

1.1 どこにあるか？

jbldarf-2.10 は圧縮された形で宇宙研の anonymous ftp サイトに置かれています。

```
ftp://ftp.astro.isas.ac.jp/asca/software/isas/jbldarf-2.10.tar.gz
```

を取ってきて展開して下さい。カレント・ディレクトリに **jbldarf-2.10/** というディレクトリができます。

1.2 コンパイルの方法

jbldarf-2.10/ の下にはいくつかのファイルと 2 つのディレクトリ (src/ と doc/) があります。まずインストールの方法が書かれている **INSTALL** を読んで下さい。この中にコンパイルの方法が書かれています。コンパイルに必要なすべてのファイルは src/ に収められています。またこのマニュアルを含む **jbldarf** に関するドキュメントが doc/ にあります。

1.3 パラメーター・ファイル

◁U▷

jbldarf は ftools と同じようにパラメーター・ファイルを持っています。パラメーター・ファイルは

```
jbldarf-2.10/jbldarf.par
```

にあります。**jbldarf-2.10** のパラメーター・ファイルは **jbldarf-2.10** より古いバージョンとの互換性がありませんので注意が必要です。

jbldarf-2.10 のパラメーター・ファイルは、展開直後は以下のようになっています。

```
Auto_build,b,h,no,,,"for automatic building"
XRT_EA_table,s,h,"/caldb/data/asca/xrt/bcf/xrt_ea_110494.fits",,,"XRT effective area table file"
XRT_PSF_type,s,h,"Table",,,"type of XRT PSF data (Analytic/Table)"
XRT_PSF_table,s,h,"/asca/caldb/xrt_psf_V1.1.fits",,,"table file of XRT point spread function"
XRT_PSF_energy_step,r,h,0,,,"Energy step in keV for XRT point spread function"
S0_teldef,s,h,"/caldb/data/asca/sis/bcf/s0_teldef_070294.fits",,,"teldef file for SIS0"
S1_teldef,s,h,"/caldb/data/asca/sis/bcf/s1_teldef_070294.fits",,,"teldef file for SIS1"
S2_teldef,s,h,"/caldb/data/asca/gis/bcf/gis2_ano_on_flf_180295.fits",,,"teldef file for GIS2"
S3_teldef,s,h,"/caldb/data/asca/gis/bcf/gis3_ano_on_flf_180295.fits",,,"teldef file for GIS3"
S2_grid_map,s,h,"/caldb/data/asca/gis/bcf/s2gridv3.fits",,,"GIS2 grid map FITS file"
S3_grid_map,s,h,"/caldb/data/asca/gis/bcf/s3gridv3.fits",,,"GIS3 grid map FITS file"
S2_Be_map,s,h,"/caldb/data/asca/gis/bcf/s2bev1.fits",,,"GIS2 Be thickness map FITS file"
S3_Be_map,s,h,"/caldb/data/asca/gis/bcf/s3bev1.fits",,,"GIS3 Be thickness map FITS file"
S2_Xe_thick,r,h,1.0,,,"GIS2 Xe thickness in cm"
```

```
S3_Xe_thick,r,h,1.0,,,"GIS3 Xe thickness in cm"
Detector_ID,i,h,0,,,"sensor ID (0-3)"
CHIPMASK_0,b,h,no,,,"mask SIS region map with Chip-0?"
CHIPMASK_1,b,h,no,,,"mask SIS region map with Chip-1?"
CHIPMASK_2,b,h,no,,,"mask SIS region map with Chip-2?"
CHIPMASK_3,b,h,no,,,"mask SIS region map with Chip-3?"
RMF_file,s,h,"default",,,"reference RMF file"
S0C0_rmf,s,h,"/caldb/data/asca/sis/cpf/94nov9/s0c0g0234p40e0_512v0_8i.rmf",,,"rmf file for S0C0"
S0C1_rmf,s,h,"/caldb/data/asca/sis/cpf/94nov9/s0c1g0234p40e0_512v0_8i.rmf",,,"rmf file for S0C1"
S0C2_rmf,s,h,"/caldb/data/asca/sis/cpf/94nov9/s0c2g0234p40e0_512v0_8i.rmf",,,"rmf file for S0C2"
S0C3_rmf,s,h,"/caldb/data/asca/sis/cpf/94nov9/s0c3g0234p40e0_512v0_8i.rmf",,,"rmf file for S0C3"
S1C0_rmf,s,h,"/caldb/data/asca/sis/cpf/94nov9/s1c0g0234p40e0_512v0_8i.rmf",,,"rmf file for S1C0"
S1C1_rmf,s,h,"/caldb/data/asca/sis/cpf/94nov9/s1c1g0234p40e0_512v0_8i.rmf",,,"rmf file for S1C1"
S1C2_rmf,s,h,"/caldb/data/asca/sis/cpf/94nov9/s1c2g0234p40e0_512v0_8i.rmf",,,"rmf file for S1C2"
S1C3_rmf,s,h,"/caldb/data/asca/sis/cpf/94nov9/s1c3g0234p40e0_512v0_8i.rmf",,,"rmf file for S1C3"
S2_rmf,s,h,"/caldb/data/asca/gis/cpf/95mar06/gis2v4_0.rmf",,,"rmf file for GIS2"
S3_rmf,s,h,"/caldb/data/asca/gis/cpf/95mar06/gis3v4_0.rmf",,,"rmf file for GIS3"
Output_file,s,h,"a.arf",,,"output ARF file"
Region_type,s,h,"Circle",,,"type of integrated region (FITS-file/Circle)"
Region_file,s,h,"reg.fits",,,"selected region file"
Center_X,r,h,0,,,"data region center x position (mm)"
Center_Y,r,h,0,,,"data region center y position (mm)"
Inner_radius,r,h,0,,,"data region inner radius (mm)"
Outer_radius,r,h,3,,,"data region outer radius (mm)"
SIS_region_rebin_X,i,h,1,,,"binning factor for SIS region map in x axis"
SIS_region_rebin_Y,i,h,1,,,"binning factor for SIS region map in y axis"
Position_type,s,h,"One-Point",,,"type of source position (FITS-file/One-Point)"
Position_file,s,h,"pos.fits",,,"source position file"
Source_X_pos,r,h,0,,,"source x position (mm)"
Source_Y_pos,r,h,0,,,"source y position (mm)"
SIS_position_rebin_X,i,h,1,,,"binning factor for SIS position map in x axis"
SIS_position_rebin_Y,i,h,1,,,"binning factor for SIS posotion map in y axis"
SIS_kobu,b,h,no,,,"Whether to multiply "kobu" factor by SIS ARF"
DEBUG,i,h,0,,,"debug flag"
clobber,b,h,no,,,"Overwrite output file if it already exists ?"
mode,s,h,"ql",,,""
```

ここでキャリブレーション・ファイルの置き場所などは宇宙研での設定になっていますので、他の機関ではリリース前に適当に書き直しておく必要があります。

宇宙研では**jbldarf-2.10.tar.gz**は展開されて

```
/adsoft/jbldarf/
```

の下に置かれていて、**jbldarf-2.10/**は単に**2.10/**とrenameされています。サンプルとなる**jbldarf.par**は

```
/adsoft/jbldarf/2.10/
```

にあります。

第 2 章

はじめての jbldarf

この章では **jbldarf-2.10** の簡単な使い方について説明します。

まず、パラメータファイルを自分の $LOCPFILES 領域にコピーして下さい。パラメータファイルがどこにあるかは、あなたがお使いの計算機に **jbldarf** をインストールした人に聞いて下さい。なお、**jbldarf-2.10** のパラメーター・ファイルは **jbldarf-2.10** より古いバージョンとの互換性がありませんので注意が必要です。

パスが正しく切ってあれば、コマンドラインから **jbldarf** と打つだけで **jbldarf** が起動できます。“CMD>” というプロンプトに続けて “show” と打つと次のようなメッセージが返されます。

```
CMD> show
                   ++=================================++
                   || jbldarf: Parameters and Status ||
                   ++=================================++

 Integrated Region
             Region type : Circle
           Center X (mm) :        0.0000000000000000
           Center Y (mm) :        0.0000000000000000
       Inner radius (mm) :        0.0000000000000000
       Outer radius (mm) :        3.0000000000000000
 Target Position
           Position type : One-Point
       Source X pos (mm) :        0.0000000000000000
       Source Y pos (mm) :        0.0000000000000000
 Map Rebinning (for SIS only)
       Region map X-axis :            1
       Region map Y-axis :            1
     Position map X-axis :            1
     Position map Y-axis :            1
 Detector ID
                  Sensor :  SIS-0
 Chip Mask (for SIS only)
        Masking chip(s) :  No Mask
 Kobu (for SIS only)
           "Kobu" factor :  NOT mutiplyed
 File Specification
           RMF file name : /caldb/data/asca/sis/cpf/94nov9/s0c1g0234p40e0_512v0_8i.rmf (default)
        Output file name : a.arf
 XRT PSF
            XRT PSF type : Table
      EA table file name : /caldb/data/asca/xrt/bcf/xrt_ea_110494.fits
     PSF table file name : /asca/caldb/xrt_psf_V1.1.fits
           Calculated at : Each energy bin
```

```
 CMD:  ********** Command **********

 Show-Parameters     Region-Type        Integrated-Region
 Position-Type       Target-Position    Map-Rebinning
 Detector-ID         Chip-Mask          Kobu-for-SIS
 File-Specification  XRT-PSF-Type       Build-ARF
 Exit                Quit

 CMD: Select  1 Option

CMD>
```

このメッセージには**jbldarf**で設定可能な全てのパラメータとそのステータスが含まれていますので、“CMD>”というプロンプトに続けて“show”と打てば、いつでも現在のパラメーターの値をチェックすることができます。**jbldarf**でユーザーが設定するのは“Integrated Region”、“Target Position”、“Detector ID”、“File Specification”、“XRT PSF”という5つの項目で、SISの場合にはこれに加えてオプションとして“Map Rebinning”、“Chip Mask”、“Kobu-for-SIS”という3つの項目を設定することができます[1]。

jbldarfで積分領域やターゲットの位置を指定する時にはDETX/DETY座標系を使います。XSELECTでXYNAMEにDETX/DETYを指定して“bin image”し、できたimageをSAOIMAGEで読めばDETX/DETY座標系でのターゲットの位置がpixel単位で得られます。**jbldarf**の入力にはこれを“mm”に変換したものを用います(セクション3.1.1)。

2.1 Integrated Region

Integrated Regionでは、ターゲットのフラックスを集積する時の積分領域の中心とその大きさを指定します。但し現時点ではまだ以下の点に注意が必要です。

XRTで点源を見た場合、各Quadrantに対応して4枚の羽ができます。実際には4枚の羽のそれぞれでエネルギー・レスポンスが異なりますが、現在のXRTの応答関数にはこの効果が含まれておらず、角度方向に積分して平均した時のみ正しい応答が得られるようになっています。**jbldarf**ではこの点を考慮し、円、または円環状の積分領域しか選べないようになっています。また積分領域の中心がターゲットの位置と同じでない場合には、出来上がったARFの精度は保証されません。XSELECTで“bin spectrum”する際にはこの点に注意して下さい。

積分領域を指定するモードに入るには、“CMD>”プロンプトに続けて“integ”と打ちます。そうするとプロンプトが“REG>”に変わります。

2.1.1 pixelからmmへの変換

ここで検出器上での積分領域の中心と外径(必要なら内径も)を“mm”単位で入力します。変換に必要なパラメーターは以下の通りです。

(a) pixelの大きさ :
0.027mm/pixel for the SIS
0.250mm/pixel for the GIS

(b) 検出器の中心座標 (DETX, DETY) :
(640.5 pixel, 640.5 pixel) for the SIS
(128.5 pixel, 128.5 pixel) for the GIS

続いてSIS、GISそれぞれについて、以下に簡単な例を示します。

---

[1] **jbldarf-0.45**までは検出器のcalibrationの目的で各detectorの光軸位置を変えることができましたが、バージョン0.50以降は“teldef”ファイルから自動的に読み込まれるようになりました。

2.1.2 SIS の場合

SAOIMAGE で読みとった積分領域の中心が (715 pixel, 678 pixel)、半径が 185 pixel の場合、

```
X: (715-640.5) * 0.027 = 2.01 mm,
Y: (678-640.5) * 0.027 = 1.01 mm,
R: (185+  0.5) * 0.027 = 5.01 mm,
```

となります。“REG>” というプロンプトに続けて

```
REG> x 2.01 y 1.01 o 5.01
```

と打てば積分領域が指定できます。もし内径が 2.00mm だったら

```
REG> x 2.01 y 1.01 i 2.00 o 5.01
```

と打てば OK です。

2.1.3 Example for the GIS

SAOIMAGE で読みとった積分領域の中心が (137 pixel, 133 pixel)、半径が 20 pixel の場合、

```
X: (137-128.5) * 0.25 = 2.13 mm,
Y: (133-128.5) * 0.25 = 1.13 mm,
R: ( 20+  0.5) * 0.25 = 5.13 mm,
```

となります。“REG>” というプロンプトに続けて

```
REG> x 2.13 y 1.13 o 5.13
```

と打てば積分領域が指定できます。もし内径が 2.00mm の場合には

```
REG> x 2.13 y 1.13 i 2.00 o 5.13
```

と打てば OK です。

2.1.4 Region Map ◁U▷

jbldarf では積分領域をイメージ FITS ファイルから読み込むことができます。このファイルを領域ファイルと呼びます。現在は領域ファイルを作るために **makemap** という tool が用意されています。

```
/adsoft/jbldarf/2.10/{ulx,osf}/makemap.
```

使い方はコマンドラインから **makemap** と打てば見ることができて、

```
usage: makemap teldef cent-x cent-y r-out [r-in] (mm)
```

となります。 GIS、 SIS それぞれについて “teldef” ファイルを指定する必要がありますが、これは

```
/caldb/data/asca/sis/bcf/s0_teldef_070294.fits
                         s1_teldef_070294.fits
                         gis2_ano_on_flf_180295.fits
                         gis3_ano_on_flf_180295.fits
```

に置かれています。従って、セクション 3.1.2 の SIS の積分領域を表す領域ファイル “region.fits” を SIS-0 について作るには

```
makemap /caldb/data/asca/sis/bcf/s0_teldef_070294.fits 2.01 1.01 5.01 2.00 > region.fits
```

とします。この region.fits は SAOIMAGE で見るとわかるように、積分領域に含まれるピクセルには 1 が、含まれないピクセルには 0 が入ったイメージ FITS ファイルです。

jbldarf にこのファイルを読ませるには以下のようにします。まず “CMD>” プロンプトに続けて “region” と入力します (Region-Type の省略)。

```
CMD> region
```

すると以下のようなメッセージが出て、プロンプトが “TYP>” に変わります。

```
TYP:  ********** Region Type **********

FITS-file Circle
TYP: Select  1 Option
TYP>
```

ここで “fits” と打って “FITS-file” の方を選んで下さい。 “CMD>” プロンプトの状態に戻って “integ” と打てば、今作った領域ファイルを入力することができます。ただし SIS の場合にはイメージのピクセル数が多いため、 ARF を作る時に、場合によっては膨大な時間がかかることがあります。このため **jbldarf** にはイメージを任意の大きさでビンまとめする機能があります (セクション 3.6)。

2.2 Target Position

ターゲットの位置を指定するには以下の二通りの方法があります。

2.2.1 One-Point

この方法は点源に対してのみ有効で、点源の位置を検出器上の座標で手で入力します。最初に show コマンドで Position type のステータスが “One-Point” になっていることを確認して下さい。もしそうでない場合には “CMD>” プロンプトが出ている状態で “position” と入力して下さい (“Position-Type” の省略形)。すると次のようなメッセージが出ます。

```
CMD> position

 TYP:  ********** Position Type **********

 FITS-file     One-Point     FFT-FITS-file XRT-PSF

 TYP: Select  1 Option
TYP>
```

ここで “one” と打って One-Point を選択します。

```
TYP> one

 CMD:  ********** Command **********

 Show-Parameters    Region-Type        Integrated-Region
 Position-Type      Target-Position    Map-Rebinning
 Detector-ID        Chip-Mask          File-Specification
 XRT-PSF-Type       Build-ARF          Exit
 Quit

 CMD: Select  1 Option
```

```
        CMD>
```

これでターゲットの位置を入力できます。“CMD>” に引き続いて “target” と入力するとプロンプトが POS> に変わります。この状態でターゲットの位置を “mm” 単位で以下のように入力します。

```
        POS> x 2.01 y 1.01
```

ピクセルから mm への変換はセクション 3.1.1 に示した通りです。

2.2.2 FITS file

積分領域を指定する場合と同様に、ターゲットの位置もイメージ FITS ファイルから入力することができます。以下ではこのファイルを「ターゲット・マップ」と呼ぶことにします。ターゲット・マップの各ピクセルは 0 ~ 1 の間の値を持っていて、**jbldarf** はターゲット位置をこの重みで平均して ARF を作ります。この性質は

1. 点源の場合には、衛星の姿勢変動に伴う視野の揺れ
2. 広がったソースの場合には、表面輝度分布

を ARF の計算に反映させるのに使われます。

まず “CMD>” プロンプトに続いて position と入力して下さい。すると

```
        CMD> position

         TYP:  ********** Position Type **********

         FITS-file     One-Point     FFT-FITS-file XRT-PSF

         TYP: Select  1 Option
        TYP>
```

というメッセージが返ってきます。ターゲット・マップに指定できるものとしては 3 種類があります。この違いについてはセクション 3.2.2、および 3.3.2を参照して下さい。ここではとりあえず “FITS-file” を選びます。

```
TYP> FITS

         CMD:  ********** Command **********

         Show-Parameters    Region-Type        Integrated-Region
         Position-Type      Target-Position    Map-Rebinning
         Detector-ID        Chip-Mask          File-Specification
         XRT-PSF-Type       Build-ARF          Exit
         Quit

         CMD: Select  1 Option
        CMD>
```

ここで target と入力すればターゲット・マップの名前を聞いてきますので、適当に入力します。

```
        CMD> target
        Source position filename: ./pos.fits ?
```

積分領域の場合と同様、SIS でターゲット・マップを読ませた場合にも計算時間が膨大になることがあります。このような場合にはターゲット・マップのピクセルを適当にビンまとめすることで時間を短縮することができます (セクション 3.6)。

2.3 Detector ID

CMD> プロンプトが出ている状態で detector と打つと detector の ID を変えることができます。選択肢は 0、1、2、3 のいずれかで、それぞれ SIS-0、SIS-1、GIS-2、GIS-3 に対応します。

2.4 File Specification

ARF を作る時に参照する RMF と出力の ARF のファイル名を指定します。CMD> プロンプトが出ている状態で file と入力します。すると次のような答が返ってきます。

```
RMF filename: default ?
```

ここで何も指定せずにリターン・キーを押すと default が選ばれ、**jbldarf.par** に書かれている RMF のうち、セクション 3.3 で指定した Detector ID に対応するものが自動的に読み込まれます[2]。これを変更したい時にはここに RMF のファイル名を書けば OK です。

続いて作られる ARF の出力先を

```
Output response filename: a.arf ?
```

というふうに聞いてきますので、適当に答えて下さい。

2.4.1 SIS の RMF

宇宙研では SIS の標準的な RMF は

```
/rukio/h1/sis/rsp1.1alpha/rsp/
```

に置かれています。ここに置かれている RMF の詳細については SIS チームの人に聞いて下さい。また自分で SIS の RMF を作るために **sis_bldrsp2** という tool が用意されています。**sis_bldrsp2** で "Disable-XRT-data" を選択し、出力として "FITS-Format" を選べば RMF を作ることができます。**sis_bldrsp2** は宇宙研では ◁U▷

```
/adsoft/sis_bldrsp2/latest/ulxbin/sis_bldrsp2   for ULTRIX,
/adsoft/sis_bldrsp2/latest/osfbin/sis_bldrsp2   for OSF/1,
```

に置かれています。詳しい使い方については README を読むか、SIS チームの人に聞いて下さい。

2.4.2 GIS の RMF ◁U▷

GIS の 1024ch の最新の RMF は

```
/caldb/data/asca/gis/cpf/95mar06/gis2v4_0.rmf
/caldb/data/asca/gis/cpf/95mar06/gis3v4_0.rmf
```

に置かれています。

2.5 XRT の PSF ◁N▷

改良された Ray-tracing プログラムによる新しい XRT の PSF が公開されています[3]。**jbldarf** ではユーザーが従来の PSF と新しい PSF を選択できるようになっています。CMD> プロンプトに続いて xrtpsf と入力すると以下のようなメッセージが返ってきて、PSF の選択ができます。

---

[2] SIS で複数のチップに跨る ARF を作る場合 (セクション 3.7)、番号の一番小さいチップに対応する RMF が読み込まれます。もしチップを選択していない場合には SIS-0、SIS-1 に対してそれぞれチップ 1、チップ 3 の RMF が読み込まれます。

[3] XRT チームによる

```
CMD> xrt-psf
 PSF:  ********** XRT PSF Type **********

 Analytic Table

 PSF: Select  1 Option

PSF>
```

ここでAnalyticというのが古いPSF、Tableが新しいPSFです。普通の解析には新しい方を使うため、tableと入力します。すると以下のように、有効面積のテーブルファイル名、PSFのテーブルファイル名、PSFを計算するエネルギーステップを順番に聞いています。

```
PSF> table
XRT effective area table file name: /caldb/data/asca/xrt/bcf/xrt_ea_110494.fits
XRT PSF table file name: /asca/caldb/xrtpsf95a.fits ?
Energy step in keV for XRT PSF calculation: 0.0 ?
```

有効面積のテーブルとPSFのテーブルは、普通の解析ではデフォルトのままでいいので空リターンします。エネルギーステップに0を指定すると、**jbldarf**はRMFと同じエネルギーステップでPSFを計算します。計算速度を上げたい場合にはここで正の数を入れて下さい。1.0を入れればPSFの計算は1.0keV刻みで行なわれます。

2.6 マップのビンまとめ (SISのみ)

領域ファイルやターゲット・マップのピクセルをまとめてからARFを計算させるには“CMD>” プロンプトにつづいてmapと入力します。すると以下のようなメッセージが返ってきますので、まとめるピクセルの数を入力します。

```
CMD> map
 MAP:  -----<< Map Rebinnig Factor (for SIS only) >>-----

 RX-Region                       1
 RY-Region                       1
 PX-Position                     1
 PY-Position                     1

 MAP: Press <ENTER> to Accept

MAP>
```

例えば領域ファイルはX/Yとも8ピクセルごとにまとめ、ターゲット・マップの方は16ピクセルごとにまとめたい場合には

```
MAP> rx 8 ry 8 px 16 py 16 ok
```

とします。もちろんX、Y別々の大きさでまとめることも可能です。

このようなビンまとめの操作のために、結果のARFが変わってくることがありますが、まとめる大きさをいろいろ変えてみて影響を調べて下さい。計算速度は当然早くなります。なお、この設定はSISのARFを作る時のみ有効で、GISのARFを作る時 (Detector IDが2または3となっている時) には何が設定してあっても無視されます。

2.7 2/4CCD モードでの ARF

2CCD モード、あるいは 4CCD モードの観測で PI コラムを使って、スペクトルを足し上げる場合の ARF の作り方を説明します。次のような 2 通りの方法があります。

2.7.1 Chip Mask

Chip Mask 機能を使うと、 One-point またはイメージ FITS ファイルによって選ばれた積分領域と、チップとのイメージの積をとることで、特定の chip のみの ARF を作ることができます。 “CMD>” プロンプトに引き続いて chip と入力して下さい。

```
CMD> chip
 MSK:  -----<< Chip Mask (for SIS only) >>-----

 No-Mask 0       1       2       3

 MSK: Select  0 or More, Then OK

MSK>
```

2CCD デフォルト位置の観測で得られた PI チャンネルのスペクトルを扱うには SIS-0 ではチップ 1 とチップ 2 を合わせた ARF が必要になります。この場合には

```
MSK> 1 2 ok
```

と入力して ARF の計算に含めるチップ (この場合は 1 と 2) を指定します。もしチップ 1 だけの ARF が欲しければ

```
MSK> 1 ok
```

とすれば OK です。同様に 4CCD モード用の ARF は

```
MSK> 0 1 2 3 ok
```

で作ることができます。 Chip Mask ではチップ間の溝による実効的な有効面積の低下を正しく評価します。

これに対して “No-Mask” を選択すると **jbldarf-0.50** 以前と同様の動作をし、チップ間の溝があることを考慮しませんので、 XRT の有効面積が額面通りに取り込まれ、結果的にソースのフラックスが小さめに出てきます。

2.7.2 Chip Maps

キャリブレーション・データー・ベースには chip map と呼ばれるイメージが入っています。これは SIS の各チップが DETX/Y 上のどの位置を占めるかを表すマップで、各チップごとに、チップにかかっているピクセルには 1 が、チップから外れているピクセルには 0 が入ったマップになっています。この chip map と **makemap** によって作られる領域ファイルの積をとることで、目的とするチップのみの ARF を作ることができます。

イメージ FITS ファイル同士に四則演算を施す tool としては **farith** があります。使い方は

```
farith <input region file> <chip map> <resulting region file> mul
```

です。ここで “mul” はイメージの対応するピクセル同士の “and” をとることを意味しています。例えば領域ファイル circle.fits のうち、 S0C1 にかかっている部分のみをとりだして result.fits という領域ファイルを作りたい時は以下のようにします。

```
farith circle.fits s0c1.fits result.fits mul
```

念のため SAOIMAGE などで result.fits を見て、目的通りの領域ファイルになっているかどうか確認して下さい。また、 Chip Map を使う時には Chip Mask のステータスが “No-Mask” になっていることを確認して下さい。

2.8 2keV 付近の「こぶ」(SIS のみ) ◁N▷

「あすか」の応答関数には、2keV 付近に原因不明の「こぶ」のような構造があることが知られています。SIS ARF を作成する時には、この構造を ARF 計算に採り入れるかどうかを選択することができます[4]。“CMD>” プロンプトにつづいて kobu と入力すると以下のようなメッセージが返ってきます。

```
CMD> kobu
Multiply "kobu" factor by SIS ARF (Y/N): N ?
```

ここで「こぶ」を ARF に採り入れたいなら、

```
Multiply "kobu" factor by SIS ARF (Y/N): N ? y
```

と入力します。「こぶ」を採り入れたくなかったら “y” の代わりに “n” と入力します。

2.9 Building the ARF response

以上の全ての設定が終ったら “CMD>” プロンプトに続けて “build” と打てば計算が始まります。もしうまく ARF ができない時や、ソフトが原因不明の異常終了をするときは是非連絡を下さい。

We Need Feedback !!

---

[4] 「こぶ」は RMF と ARF のどちらか一方にだけ入っているのが正しい状態です。ARF に「こぶ」を採り入れた場合には「こぶ」の入っていない RMF を、ARF に「こぶ」を採り入れなかった場合には「こぶ」の入っている RMF を、それぞれ組み合わせる必要があります。

第 3 章

jbldarf-2.10 を正しく使うために

本章は **jbldarf** をよりよく理解し、活用したい人のための解説です。

jbldarf は、いろいろな観測条件や解析方法について、対応するレスポンスを計算することができます。しかし、「あすか」のレスポンスはまだまだ開発途上であり、現状では不十分な点も多いと感じています (§3.4参照)。「あすか」の解析で **jbldarf** を使われる方には、ぜひとも、本章を一読なさって、**jbldarf** の特長や弱点を踏まえた上で使ってほしいと思います。

3.1 ARF ファイルとは

3.1.1 XSPEC とレスポンス

jbldarf がどのような計算を行なうかを見る前に、簡単にレスポンスについておさらいしておこう。スペクトルフィットを行なう場合、XSPEC はモデル関数 $\mathrm{M}[E]$、RMF レスポンス $\mathrm{R}[PHA,E]$、および ARF レスポンス $\mathrm{A}[E]$ から検出器での検出が予想される PHA スペクトル $\mathrm{S}[PHA]$ を次のように計算する[1]。

$$\mathrm{S}[PHA] = \sum_{E} \mathrm{R}[PHA,E]\ \mathrm{A}[E]\ \mathrm{M}[E]$$

XSPEC は、この $\mathrm{S}[PHA]$ と観測された PHA スペクトルが合うようにモデル関数のパラメータを調整する[2]。

レスポンスをこのように 2 つにわける利点は、レスポンスのうち観測や解析の方法に依存しない部分や、センサ間で共通な部分を共有することができる点にある[3]。こうすることで、レスポンスの計算にかかる時間を短縮でき、さらにディスク領域も節約することができる。

余談になるが、上の式では、いわゆる検出器の応答関数に相当するものは $\mathrm{R}[PHA,E]\ \mathrm{A}[E]$ という積の形で表現されているため、RMF レスポンスと ARF レスポンスの定義の方法は一意的でない[4]。実際には、SIS の場合、$\sum_{PHA} \mathrm{R}[PHA,E]$ はエネルギー E での検出効率に相当し、ARF は有効面積の部分を受け持っている。GIS の場合には、$\sum_{PHA} \mathrm{R}[PHA,E]$ は 1 に規格化されており[5]、ARF が有効面積と検出効率の部分を受けもつようになっている。

3.1.2 ARF ファイルの構造

レスポンスが思った通りに作成できたかどうかを調べるには、ARF ファイルをのぞいてみるのが一番である。この節では、そのために必要な知識をまとめておこう。

---

[1] そのはずである。確かめたわけではない。

[2] 実際には、バックグラウンドの差し引きなども行なわれる。

[3] 実際、現状では GIS の RMF レスポンスは、GIS2 と GIS3 で同じものを使用している。

[4] 例えば、RMF レスポンスを 2 倍にするのと同時に ARF レスポンスを 2 分の 1 にすれば、結果として同じ応答関数を得る。

[5] スペクトルが RMF のエネルギーレンジからはみ出す場合もあるので、常に 1 になるわけではない。

ARF ファイルは FITS 形式のファイルで最低 1 つの BINTABLE extension を持つ。fstruct[6]でみてみると、例えば次のようになっているはずである。

```
No. Type     EXTNAME      BITPIX Dimensions(columns)      PCOUNT  GCOUNT
  0  PRIMARY                 8     0                          0     1
  1  BINTABLE SPECRESP       8     20(5) 201                  0     1
```

EXTNAME が 'SPECRESP' となっているところが ARF の本体である[7]。jbldarf の場合は、1st extension に BINTABLE を作るようになっている。

さて、その BINTABLE には、最低でも次の 3 つの項目があるはずである。

ENERG_LO, ENERG_HI, SPECRESP

ENERG_LO、ENERG_HI はそのビンのエネルギー範囲を決め、SPECRESP がそのエネルギー範囲での A[E] を与える。jbldarf はその他に、**EFFAREA, EXPOSURE** という項目も ARF ファイルに書き出すようになっている。EFFAREA には目安としての有効面積が、EXPOSURE には SPECRESP を EFFAREA で割ったものが入っている。いずれも、XSPEC では参照されないのでスペクトルフィットには無関係だが、つくられた ARF のだいたいの様子を知りたい時には便利である。fplot[8]で一度それらの項目をプロットしてみるといいだろう。

3.1.3 ヘッダのキーワード ◁U▷

jbldarf は、ARF ファイル作成時のさまざまな情報を ARF ファイルのヘッダに書き出すようになっている。レスポンスが何か変だと思った時は、ヘッダを注意深く見てみれば何が悪かったのかわかることが多い。

以下に jbldarf が ARF に書き出すヘッダの主なものの一覧とその内容を示す。なお、以下では、座標軸毎に書き出されるものは ???X(Y) などとして一括して表記してある。また、右肩に 2.00 とついているキーワードは、jbldarf-2.00 から新しく導入されたものである。

プライマリヘッダ ◁U▷

DATE	この ARF ファイルを作った日 / 月 / 年
CREATOR$^{2.00}$	この ARF ファイルを作った人のログイン名[9] ◁U▷
AUTHOR	jbldarf のバージョン
ORIGIN	jbldarf working group

SIS/GIS 共通のヘッダ ◁U▷

NAXIS2	エネルギーのビン数。参照した RMF と同じ数になる。
TELESCOP	通常 'ASCA'。参照した RMF からコピーする。
INSTRUME	通常 'SIS0' 'SIS1' 'GIS2' 'GIS3' のいずれか。参照した RMF からコピーする。
DET_ID	ARF を作った時の Detector-ID。0 から 3 の値をとる。
TELDEF	検出器のアライメント情報を引っ張ってきた teldef ファイル[10] の名前。
XRT_EA	XRT の有効面積を引っ張ってきたテーブルのファイル名。
PSFTYPE $^{2.00}$	XRT の Point Spread Function の計算方法。'Analytic'、'Table' のいずれか。 ◁N▷
PSFTABLE $^{2.00}$	XRT の Point Spread Function を引っ張ってきたテーブルのファイル名。PSF-type が 'Table' の時のみ書かれる。 ◁N▷

---

[6] FTOOLS のコマンドのひとつで、FITS ファイルの構造を調べるのに役立つ。

[7] jbldarf-0.5 より前のものでは 'ARF' となっている。

[8] FTOOLS のコマンドのひとつ。FITS のテーブルをグラフにできる。

[9] jbldarf-2.00 より前は CREATER だったので注意。

[10] *Telescope definition* の略。もともと ASCALIN 用のキャリブレーションファイル。jbldarf では、XRT 座標、検出器ミリメートル座標、検出器チャンネル座標の相互変換のために使用する。

PSFESTEP $^{2.00}$	XRT の Point Spread Function を計算するエネルギーの刻み。単位は keV。 PSF-type が 'Table' の時のみ書かれる。 ◁N▷
FC_X(Y)_SCL	検出器上での X(Y) 方向角度スケール (arcmin/mm)。 teldef ファイルからコピーされる。X/Y で値が違っていた場合の計算結果は保証できない。
OPTAXISX(Y)	検出器ミリメートル座標での光軸の X(Y) 位置。 teldef ファイルにも同様なキーワードがあるが、衛星座標系で書かれているので注意。
DET_X(Y)CEN	検出器チャンネル座標の X(Y) 方向の中心 (ミリメートル座標が 0 となる位置)。 teldef ファイルからコピーされる。
DET_X(Y)SCL	検出器上での X(Y) 方向ピクセルサイズ (mm/pixel)。 teldef ファイルからコピーされる。X/Y で値が違っていた場合の計算結果は保証できない。
DETX(Y)PIX1	検出器チャンネル座標系での X(Y) 方向の最小ピクセル番号。 1 で固定。
REGTYPE	ARF 作成時の Region-type。'Circle'、'FITS-file' のいずれか。
REGCENTX(Y)	リージョン中心の X(Y) 座標 (mm)。 Region-type が 'Circle' の時のみ書かれる。
REGRIN	リージョンの内半径 (mm)。 Region-type が 'Circle' の時のみ書かれる。
REGROUT	リージョンの外半径 (mm)。 Region-type が 'Circle' の時のみ書かれる。
REGFILE	Region-file の値。 Region-type が 'FITS-file' の時のみ書かれる。
POSTYPE	ARF 作成時の Position-type。'One-Point'、'FITS-file'、'FFT-FITS-file'、'XRT-PSF' のいずれか。
POSCENTX(Y)	検出器ミリメートル座標での X(Y) 方向ソース位置。 Position-type が 'One-Point' の時のみ書かれる。
POSFILE	Position-file の値。 Position-type が 'FITS-file' か 'FFT-FITS-file' か 'XRT-PSF' の時のみ書かれる。
XRT_OFF	ソース位置の XRT の光軸からの離角 (arcmin)。
XRT_AZM	ソース位置の XRT の光軸からの方位角 (degree)。衛星座標上で定義されているので、検出器座標では X 軸の正方向から時計回りに測られるので注意。

SIS 専用のヘッダ

◁U▷

REGRBINX(Y)	リージョンマップの X(Y) 方向のビンまとめ数。
POSRBINX(Y)	ポジションマップの X(Y) 方向のビンまとめ数。 Position-type が 'FITS-file' の時のみ書かれる。
CHIPMASK	リージョンマップのマスクに用いたチップの CCD ID (例:`'0 1 2 3'`)。マスクしなかった時は`'No Mask'` となる。
SIS_KOBU$^{2.10}$	2keV 付近の「こぶ」を ARF 計算に採り入れたかどうか。採り入れたら'Multiplied'、採り入れなかったら'NOT multiplied'。

GIS 専用のヘッダ

REGPIXLS	$\int d\vec{r}_3 w_{\rm R}(\vec{r}_3)$ の値 (§3.3.1)。要するに、リージョンの面積 (pixels)。
INTG_X(Y)0	計算を行なった領域の X(Y) 方向の最小ピクセル番号。ただし、ピクセル番号は 0 から始まっている。
INTG_NX(Y)	計算を行なった領域の X(Y) 方向の大きさ (pixel)。
POSRSIZE	ポジションマップの広がり (pixel)。 Position-type が 'FITS-file' か 'FFT-FITS-file' か 'XRT-PSF' の時のみ書かれる。

GRIDFILE	井桁マップのファイル名。
GRIDTRNS	リージョンでの井桁マップの平均透過率 (%)。
BE_FILE	Be 膜厚マップのファイル名。
BE_THICK	リージョンでの Be 膜厚の平均値 (μm)。
XE_THICK	ドリフト領域の Xe の厚さ (cm)。
RSPVERSN	GIS のレスポンス関数のバージョン。

3.2 何を計算するのか (SIS 編)

3.2.1 計算式

◁U▷

jbldarf が計算していることは、次のような式で書くことができる[11]。

$$\mathrm{A}[E] = F_{\mathrm{kobu}}(E)T_{\mathrm{xrt}}(E)\int d\vec{r}_1 w_{\mathrm{P}}(\vec{r}_1)g_{\mathrm{xrt}}(E;\vec{r}_1)\int d\vec{r}_3 w_{\mathrm{R}}(\vec{r}_3)P_{\mathrm{xrt}}(E;\vec{r}_3,\vec{r}_1)$$

各記号の意味は、次のようになっている。

$P_{\mathrm{xrt}}(E;\vec{r}_3,\vec{r}_1)$	XRT の *Point Spread Function,* すなわち、 $\vec{r}_1$ 方向から X 線が来た時に、検出器上で $\vec{r}_3$ の位置に反射される確率。
$g_{\mathrm{xrt}}(E;\vec{r}_1)$	$\vec{r}_1$ 方向からエネルギー E の X 線が来た時の XRT の *Effective Area*。
$w_{\mathrm{P}}(\vec{r}_1)$	*Position Map,* すなわち、 $\vec{r}_1$ 方向にソースが存在する確率。 $\vec{r}_1$ で積分した値が 1 になるように規格化されている。
$w_{\mathrm{R}}(\vec{r}_3)$	*Region Map,* すなわち、 $\vec{r}_3$ の位置に検出されたイベントを採用する確率。
$T_{\mathrm{xrt}}(E)$	XRT に施された Thermal Shield の X 線透過率。
$F_{\mathrm{kobu}}(E)$	2keV 付近の「こぶ」を表した因子。中心約 2keV、幅約 0.1keV のガウシアンに 1.0 を加えたもの。

式を見てわかるように、 $g_{\mathrm{xrt}}(E;\vec{r}_1)$ と $P_{\mathrm{xrt}}(E;\vec{r}_3,\vec{r}_1)$ は特に分ける必要はないのだが、プログラムの便宜上、分けられている。また、 $\vec{r}_1, \vec{r}_3$ は、 X 線が来る方向[12]、実際に検出された位置 (SIS の場合、検出器面上で X 線が来た位置に一致する[13]) にそれぞれ対応する。積分領域は、計算式の上では全領域に対してということになるが、実際には、 $w_{\mathrm{R}}(\vec{r}_3)$ が 0 でない領域の近辺でのみ積分を行なっている。

3.2.2 より具体的には

◁U▷

もっと具体的に計算の流れを書くと、以下のようになる。

(1) ポジションマップを準備する。

  (a) Position-type が **'One-Point'** のとき
  $\vec{r}_1$ = (Source-X-pos, Source-Y-pos) をセットする。

  (b) Position-type が **'FITS-file'** のとき
  Position-file を読み込み、指定されたビンまとめを行ない、さらに、足して 1 になるようあらかじめ規格化する。格子点の数が多いと、後の計算で非常に時間がかかって使用に耐えないので、解析の主旨や XRT の性能などを十分に考慮してビンまとめ数を指定する必要がある (3.4.4参照)。

---

[11] ただし、 $F_{\mathrm{kobu}}(E)$ は、 ARF 計算に「こぶ」を採り入れる時のみ乗ずる。

[12] 天球の絶対的な方向ではなく、あくまで XRT に対する相対的な方向であることに注意。

[13] このことは、検出器の *Point Spread Function* がデルタ関数的であることに対応している。

(2) リージョンマップを準備する。

まず、以下のように、もとになるマップを作る。

(a) Region-type が **'Circle'** のとき
リージョンの中心や半径などに従ってマップを作り、メモリ上に保持する。

(b) Position-type が **'FITS-file'** のとき
Region-file を読み込み、メモリ上に保持する。

次に、指定されたチップの位置にマスクをかける。すなわち、指定されたチップの上にない $\vec{r}_3$ に対して $w_{\rm R}(\vec{r}_3)=0$ をセットする。[14]チップマスクが**'No-Mask'** のときには何もしない。[15]

最後に、指定されたビンまとめを行なう。ここでも、格子点の数が多いと、後の計算で非常に時間がかかって使用に耐えないので、解析の主旨や XRT の性能などを十分に考慮してビンまとめ数を指定する必要がある (3.4.4参照)。

(3) ソースポジション $\vec{r}_1$ に対する $g_{\rm xrt}(E;\vec{r}_1)$ を計算する。
$\vec{r}_1$ に対する $g_{\rm xrt}(E;\vec{r}_1)$ をすべてのエネルギーについてまとめて計算し、メモリ上に保持する。これ以降の計算ではメモリ上の値を参照する。このような操作を行なうのは計算を高速化するためである。

(4) $w_{\rm P}(\vec{r}_1)g_{\rm xrt}(E;\vec{r}_1)\int d\vec{r}_3 w_{\rm R}(\vec{r}_3)P_{\rm xrt}(E;\vec{r}_3,\vec{r}_1)$ を計算し、 A$[E]$ に加算する。
積分計算の部分はループを回して足し算をするだけである。ただし、この足し算は、すべてのエネルギーに対して並列に行なう。このとき、前のステップで計算しておいた $g_{\rm xrt}(E;\vec{r}_1)$ をメモリから読み出して用いる。 A$[E]$ への加算は $\vec{r}_1$ 積分の 1 ステップに対応する。

なお、計算速度の向上のため、 PSF の計算は、以下のように、特定のエネルギーでのみ行なっている。 ◁N▷

(a) PSF-type が 'Analytic' のとき
供給されている XRT の PSF がエネルギーに依存しない形に書かれていることを利用し、エネルギーごとの PSF の計算を省略する。つまり、すべてのエネルギーに対して、始めに一度だけ PSF が計算される。

(b) PSF-type が 'Table' のとき
ユーザの指定に応じたエネルギー刻みで、 PSF を計算する。例えば、エネルギー刻みとして 1keV を指定した場合、 0.5keV、 1.5keV、 2.5keV、 $\cdots$ で PSF を計算し、それぞれ、 0~1keV、 1~2keV、 2~3keV、 $\cdots$ における ARF の計算に用いる。

(5) $\vec{r}_1$ を変えて、 (3) へ戻る。

(6) すべての $\vec{r}_1$ について計算し終ったら、 $T_{\rm xrt}(E)$ を掛ける。
加算された A$[E]$ には、 $\int d\vec{r}_1 w_{\rm P}(\vec{r}_1)g_{\rm xrt}(E;\vec{r}_1)\int d\vec{r}_3 w_{\rm R}(\vec{r}_3)P_{\rm xrt}(E;\vec{r}_3,\vec{r}_1)$ の計算結果が入っている。
そこで、これにエネルギー毎に $T_{\rm xrt}(E)$ を掛ける。

(7) 「こぶ」を採り入れるよう指定された時には、 $F_{\rm kobu}(E)$ を掛ける。
この時点で A$[E]$ には、 $T_{\rm xrt}(E)\int d\vec{r}_1 w_{\rm P}(\vec{r}_1)g_{\rm xrt}(E;\vec{r}_1)\int d\vec{r}_3 w_{\rm R}(\vec{r}_3)P_{\rm xrt}(E;\vec{r}_3,\vec{r}_1)$ の計算結果が入っている。そこで、これにエネルギー毎に $F_{\rm kobu}(E)$ を掛け、最終的な A$[E]$ を得る。

---

[14] この操作によって、視野外や溝の部分は常に $w_{\rm R}(\vec{r}_3)=0$ となり、 ARF の計算から除外される。

[15] したがって、この場合、視野の外や溝の部分も (リージョンマップに含まれていれば) ARF に加算される。

3.2.3 DEBUG パラメータ

jbldarf の DEBUG パラメータを ON にすると、計算の途中結果のイメージをカレントディレクトリの ./map/ というディレクトリに書き出すようになっている[16]。書き出されるイメージには、次のようなものがある。

original_posmap.fits	ビニング前の *Position Map*
binned_posmap.fits	ビンまとめされた *Position Map*
original_region.fits	チップマスク前の *Region Map*
masked_region.fits	チップマスクされた *Region Map*
binned_region.fits	ビンまとめされた *Region Map*

実際の計算に用いられた $w_{\rm R}(\vec{r}_3)$ と $w_{\rm P}(\vec{r}_1)$ は、それぞれ、binned_regmap.fits、binned_posmap.fits である。これらを saoimage などで見れば、ビンまとめが適切であったかどうかを確認することができる。

3.3 何を計算するのか (GIS 編)

3.3.1 計算式

jbldarf が計算していることは、次のような式で書くことができる。

$$\mathrm{A}[E] = \int d\vec{r}_3 w_{\rm R}(\vec{r}_3) \int d\vec{r}_2 g_{\rm gis}(E;\vec{r}_2) P_{\rm gis}(E;\vec{r}_3,\vec{r}_2) \int d\vec{r}_1 w_{\rm P}(\vec{r}_1) g_{\rm xrt}(E;\vec{r}_1) P_{\rm xrt}(E;\vec{r}_2,\vec{r}_1)$$

各記号の意味は、次のようになっている。

$P_{\rm xrt}(E;\vec{r}_2,\vec{r}_1)$	XRT の *Point Spread Function,* すなわち、$\vec{r}_1$ 方向から X 線が来た時に、検出器上で $\vec{r}_2$ の位置に反射される確率。
$g_{\rm xrt}(E;\vec{r}_1)$	$\vec{r}_1$ 方向からエネルギー E の X 線が来た時の XRT の *Effective Area*。
$w_{\rm P}(\vec{r}_1)$	*Position Map,* すなわち、$\vec{r}_1$ 方向にソースが存在する確率。$\vec{r}_1$ で積分した値が 1 になるように規格化されている。
$P_{\rm gis}(E;\vec{r}_3,\vec{r}_2)$	GIS の *Point Spread Function,* すなわち、検出器上で $\vec{r}_2$ の位置に X 線が来た時に、その位置が $\vec{r}_3$ として検出される確率。
$g_{\rm gis}(E;\vec{r}_2)$	検出器上で $\vec{r}_2$ の位置に X 線が来た時の GIS の検出効率。
$w_{\rm R}(\vec{r}_3)$	*Region Map,* すなわち、$\vec{r}_3$ の位置に検出されたイベントを採用する確率。

式を見てわかるように、$g_{\rm xrt}(E;\vec{r}_1)$ と $P_{\rm xrt}(E;\vec{r}_2,\vec{r}_1)$、あるいは $g_{\rm gis}(E;\vec{r}_2)$ と $P_{\rm gis}(E;\vec{r}_3,\vec{r}_2)$ は特に分ける必要はないのだが、プログラムの便宜上、分けられている。また、$\vec{r}_1,\vec{r}_2,\vec{r}_3$ は、X 線が来る方向[17]、検出器面上で X 線が来た位置、実際に検出された位置、にそれぞれ対応する。積分領域は、計算式の上では全領域に対してということになるが、実際には、$w_{\rm R}(\vec{r}_3)$ が 0 でない領域の近辺でのみ積分を行なっている。

3.3.2 より具体的には ◁U▷

もっと具体的に計算の流れを書くと、以下のようになる。

(1) $psf(E;\vec{r}_2) = \int d\vec{r}_1 w_{\rm P}(\vec{r}_1) g_{\rm xrt}(E;\vec{r}_1) P_{\rm xrt}(E;\vec{r}_2,\vec{r}_1)$ を計算する。

  (a) Position-type が **'One-Point'** のとき
  $\vec{r}_1$ = (Source-X-pos, Source-Y-pos) での $g_{\rm xrt}(E;\vec{r}_1) P_{\rm xrt}(E;\vec{r}_2,\vec{r}_1)$ のみを計算する。

---

[16] ./map/ ディレクトリがなければ作らない。

[17] 天球の絶対的な方向ではなく、あくまで XRT に対する相対的な方向であることに注意。

(b) Position-type が **'FITS-file'** のとき
$w_{\mathrm{P}}(\vec{r}_1)$ が 0 でない格子点で $g_{\mathrm{xrt}}(E;\vec{r}_1)P_{\mathrm{xrt}}(E;\vec{r}_2,\vec{r}_1)$ を計算し、$w_{\mathrm{P}}(\vec{r}_1)$ の重みをつけて足し合わせる。$w_{\mathrm{P}}(\vec{r}_1)$ は Position-file から読み込まれ、足して 1 になるようあらかじめ規格化されている。格子点の数が多い時は、非常に遅くなって、使用に耐えない。

(c) Position-type が **'FFT-FITS-file'** のとき[18]
$w_{\mathrm{P}}(\vec{r}_1)$ の重心で $P_{\mathrm{xrt}}(E;\vec{r}_2,\vec{r}_1)$ を計算し、$w_{\mathrm{P}}(\vec{r}_1)g_{\mathrm{xrt}}(E;\vec{r}_1)$ との convolution をとる。convolution には FFT を用いるので、上の場合より一般に速くなる。

(d) Position-type が **'XRT-PSF'** のとき[19]
FITS ファイルから $psf(E;\vec{r}_2)$ を読み込む。詳しくは、§3.5.3 を参照のこと。

なお、計算速度の向上のため、PSF の計算は以下に示す特定のエネルギーでのみ行なわれる。 ◁N▷

(i) PSF-type が 'Analytic' のとき
供給されている XRT の PSF がエネルギーに依存しない形に書かれていることを利用し、エネルギーごとの PSF の計算を省略する。つまり、すべてのエネルギーに対して、始めに一度だけ PSF が計算される。

(ii) PSF-type が 'Table' のとき
ユーザの指定に応じたエネルギー刻みで、PSF を計算する。例えば、エネルギー刻みとして 1keV を指定した場合、0.5keV、1.5keV、2.5keV、$\cdots$ で PSF を計算し、それぞれ、0～1keV、1～2keV、2～3keV、$\cdots$ における ARF の計算に用いる。

(2) $eff(E;\vec{r}_2) = g_{\mathrm{gis}}(E;\vec{r}_2)psf(E;\vec{r}_2)$ を計算する。
$g_{\mathrm{gis}}(E;\vec{r}_2)$ は、Be 膜厚マップと井桁マップから計算される。$\vec{r}_2$ での Be 膜厚マップの値から、そこでの GIS の検出効率[20]を計算し、それに井桁マップの値と、(1) で計算した $psf(E;\vec{r}_2)$ が掛け算されて、$eff(E;\vec{r}_2)$ ができ上がる。

(3) $expo(E;\vec{r}_3) = \int d\vec{r}_2 P_{\mathrm{gis}}(E;\vec{r}_3,\vec{r}_2)eff(E;\vec{r}_2)$ を計算する。
通常使用される範囲では、GIS の位置分解能はあまり変化しない。そこで、ソース位置での $P_{\mathrm{gis}}(E;\vec{r}_3,\vec{r}_2)$ と $eff(E;\vec{r}_2)$ の FFT を用いた convolution を行なうことで、計算時間を短縮している。

(4) $\mathrm{A}[E] = \int d\vec{r}_3 w_{\mathrm{R}}(\vec{r}_3)expo(E;\vec{r}_3)$ を計算する。
これはもう、ループを回して足し算をするだけである。

(5) E を変えて、(1) へ戻る。

3.3.3 DEBUG パラメータ

jbldarf の DEBUG パラメータを ON にすると、計算の途中結果のイメージをカレントディレクトリの ./map/ というディレクトリに書き出すようになっている[21]。書き出されるイメージには、次のようなものがある。

---

[18] GIS のみ有効。

[19] これも、GIS のみ有効。

[20] ここでは Be 膜の透過率だけでなく、Xe ガスの吸収係数、サーマル / プラズマシールドの透過率も計算される。

[21] ./map/ ディレクトリがなければ作らない。

bemap.fits	Be 膜厚マップ
effnnn.fits	$\mathit{eff}(E;\vec{r}_2)$
exponnn.fits	$\mathit{expo}(E;\vec{r}_3)$
gispsfnnn.fits	$P_{\rm gis}(E;\vec{r}_3,\vec{r}_2)$
grid.fits	井桁マップ
posmap.fits	*Position Map*
region.fits	*Region Map*
xrtpsf.fits	$\mathit{psf}(E;\vec{r}_2)$

nnn がつくものは、エネルギーの刻みごとにつくられる。xrtpsf.fits が 1 つしか作られないのは、現状の XRT レスポンスでは、エネルギーによって *Point Spread Function* が変化しないからである[22]。また、これらのイメージは、実際に計算が行なわれる範囲に限って書き出される。書き出された範囲は ARF のヘッダに書いてある。

3.4 適用限界

jbldarf の適用限界は、大まかに分けて、提供されているレスポンスの信頼性と、jbldarf での ARF の計算方式とに起因すると言える。さらに、提供されているレスポンスは XRT 部と GIS 部に分けられ[23]、$g_{\rm xrt}(E;\vec{r}_1)$ と $P_{\rm xrt}(E;\vec{r}_2,\vec{r}_1)$ が XRT 部、$g_{\rm gis}(E;\vec{r}_2)$ と $P_{\rm gis}(E;\vec{r}_3,\vec{r}_2)$ が GIS 部に属する。以下では、これらの関数について概説し、ARF を作る上での問題点を述べる。

3.4.1 XRT のレスポンスの観点から ◁U▷

jbldarf が使用している XRT のレスポンスは、有効面積を与えるものと、PSF を与えるものの 2 種類である。いずれも、FORTRAN の関数で、有効面積を与えるものは、

```
Real Function xrt_eff_area(e_min, e_max, theta, phi)
Real e_min, e_ma, theta, phi
```

PSF を与えるものは、

(a) PSF-type が 'Analytic' のとき

```
Real Function xrtpsf(E,x1,y1,x2,y2)
Real E, x1, y1, x2, y2
```

(b) PSF-type が 'Table' のとき ◁N▷

```
Real Function xrtpsf95a(E,theta,phi,ox,oy)
Real*8 E, theta, phi, ox, oy
```

という形になっている。

現在の jbldarf で使用しているものでは `xrt_eff_area()` はパラメータファイルで指定された FITS の有効面積テーブル (XRT-EA-table) から読み込んだものを適当に補間して返すようになっていて、信頼性はそのテーブルファイルによる。関数形を見てわかるように、センサー ID を渡すことはできず、センサーごとの個性は考慮していない。XRT チームによると、現状の XRT-EA-table には Ray-tracing によるデータを元に、次の範囲のデータが入っている。

名前	単位	形式	範囲 (刻み)
Energy	keV	1E	0.1–2.0 (0.1), 2.01–4.00 (0.01), 4.1–12.0 (0.1)
Theta	arcmin	1I	0–20 (1)
Phi	degree	1I	0–45 (15)
EA	cm^2	1E	—

---

[22] Position-type に 'XRT-PSF' を選んだ時はその限りではない。

[23] SIS のレスポンスはすべて RMF に吸収されているため、ARF の計算には使われない。

当然、この範囲を越えた指定をした場合のレスポンスは保証されない。

PSF-type が 'Analytic' のときに用いられる `xrtpsf()` は、 XRT の PSF を内部で適当な算術関数で近似して計算を行なっている。これも、 FORTRAN の関数形を見てわかるように、センサーごとの個性は考慮していない。また、現在使用しているものでは「ちょうちょ」とか「よつばのクローバー」とかよばれる形は入っておらず[24]、端にいくと PSF が歪む効果も入っていないようである。さらに、引数にはエネルギーが入っているが、関数の中では全く使われておらず、エネルギーの依存性は有効面積の方に任せた形になっている。ついでにいっておけば、ピークもちょっと鋭すぎて、現実と合わないようだ。以上のことから、リージョンをソース位置を中心とした円以外の形で切ったり[25]、リージョンの半径を大きくとりすぎたり小さくとり過ぎたり[26]した場合の ARF は信頼度が低くなる。

◁N▷ PSF-type が 'Table' のときに用いられる `xrtpsf95a()` は、 Ray-tracing を用いて計算した PSF のテーブルファイルを読み込み、一次補間を行なって ARF 計算に必要な位置での PSF を求めている。これもセンサーごとの個性は考慮していないが、現時点での最新の Ray-tracing の結果が反映されているため、 PSF のエネルギー依存性や、いわゆる「つの」の効果も採り入れられている。ただし、テーブルは、 EEF(Encircled Energy Function) の形で供給されているので、ソース位置を中心とした円形領域以外でスペクトルを切りとったりした場合、 ARF は信頼度が低くなる。

まとめると XRT のレスポンスの観点から気をつけるべきことは、次のようなことである。

(1) エネルギーは 0.1 – 12 keV、ソース位置は光軸から 20 arcmin 以内。
(2) リージョンはソース位置を中心とした円または円環で切る。
(3) リージョンの半径は 1.5 – 6 arcmin 程度。

◁N▷ ただし、これらの値は、今後 XRT の有効面積テーブルや PSF テーブルが改訂されれば、それにともなって改善する可能性がある。両テーブルの適用限界については XRT チームから発表されるものを参照すること。

3.4.2 GIS のレスポンスの観点から

jbldarf が使用している GIS のレスポンスは C の関数で、

```
double get_gis_eff(double e_lo, double e_hi);
double gis_psf(double xmm, double ymm, double Ekev, double xoff, double yoff);
```

があり、それぞれ GIS の検出効率と PSF を与えるものである。関数形からわかるように、センサーごとの違いは考慮されていない。また、井桁マップや Be 膜厚マップもレスポンスを決める重要な要素である。

`get_gis_eff()` では、 XRT のサーマルシールド、プラズマ防御膜、 Be 膜、サポートメッシュ[27]の透過率と、 Xe+He ガスの吸収効率を計算している。エネルギーの範囲は 0.5–15keV くらいまでを想定している。位置による違いは入っていないが、特に問題ないと思われる。

`gis_psf()` は、実験のデータと、 XRT の入射角のばらつきによる広がりを考慮して、中心対称な 2 次元ガウシアンの重ね合わせとして PSF を計算している。中心から 20mm より外では位置分解能が悪くなる効果も入っているが、 3.3.2で述べたように、 jbldarf の計算ではソース位置での PSF しか使っていないので、あまり意味はない。したがって、センサーの周辺付近ではレスポンスの信頼度が落ちる。また、 GIS の位置分解能はエネルギーによって違うため、エネルギーによってリージョンの外に洩れ出すフォトンの数は違ってくることになる。当然、リージョンが小さい方がこの効果が大きくなるが、これがどこまで正確に再現できているかについては、はっきりとは調べられていない。したがって、リージョンをあまり小さくとると、 ARF の信頼度は落ちる。

Be 膜厚マップは、平均的な厚さは地上実験を元にしたデータが入っているが、位置によるばらつきに関してはあまり精度は良くない。しかし、位置によるばらつきは、最大で 0.6μm(GIS-2) / 0.4μm(GIS-3) 程度 (上田修論、 1993) であり、それほど大きな影響はないと考えられる。

---

[24] 試しに適当な位置での PSF のイメージを作らせてみるとわかる。そのために、./src/gis/xrtpsf.c というプログラムが用意されている。デフォルトではコンパイルされない。

[25] 「ちょうちょ」や PSF の歪みが効いてくると思われる。

[26] 中心付近や tail 付近では、 PSF が正しく再現されていない。

[27] 0.87 でエネルギーによらず固定となっている。

井桁マップは、位置や透過率がエネルギーによらないことになっているが、これが本当かどうかを調べるのは統計的に苦しく、よくはわかっていない。また、井桁の位置は jbldarf の仕様上も、現実的にも、1pixel の精度となっている (松下修論、1994)。ソースが井桁のすぐそばにあった時の影響についてはまだ良くは調べられておらず、jbldarf がそれをちゃんと再現できるかどうかは怪しい。従って、井桁付近にソースがある時の ARF も信頼度は低い。

まとめると GIS のレスポンスの観点から気をつけるべきことは、次のようなことである。

(1) エネルギーは 0.5 – 15 keV。
(2) センサーの周辺部 ($\sim$ 20mm より外) は避ける。
(3) リージョンの半径は $\sim$ 1.5 arcmin より大きく。
(4) 井桁のそばのソースは避ける。

3.4.3 jbldarf の構造に起因する限界

現状の jbldarf では、以下のような場合に対応する ARF を作るのは困難である。

(1) 視野外にまでソースが広がっている場合。

Position-file は視野内しかカバーしていないため、視野外からの洩れ込みの影響は考慮されない。Position-type を'XRT-PSF' にして、あらかじめその効果を計算しておけば、取り込むことは可能である。

(2) 位置によりスペクトルが違っている場合。

そもそも XSPEC 自体が、位置ごとにスペクトルが違う時のフィットをできるようになっていない[28]。このような解析をする時は、スペクトルが同じだと思える範囲でリージョンを切り分けたり、エネルギーバンドごとにイメージを作ったりして攻めるのが適当であろう。

3.4.4 SIS のマップのビンまとめ

SIS の ARF の計算式 (3.2.1参照) に現れる積分は、実際の計算では、SIS の 1 ピクセルを単位として数値積分される (ビンまとめしない場合)。ところが、計算速度を測定してみると、通常の解析においては、SIS の ARF の計算にかかる時間はほとんどピクセル数のみが支配していることがわかる。例えば、円形のリージョンマップを使っている場合、半径を 2 倍にすると計算にかかる時間は 4 倍になる。広がったソースの場合には、有限の広がりを持つポジションマップを使わざるを得ないため、ポジションマップの非零のピクセル数が N 個の場合には、計算時間はポイントソースの場合の N 倍かかってしまい、実用に耐えない。

そこで jbldarf では、計算速度向上のため、リージョンマップとポジションマップのそれぞれについて、指定された数だけビンまとめしてから計算するようにしている。この場合に気をつけなければならないのが、ビンまとめ数をいくつに設定すべきか、という点である。

まず、リージョンマップのビンまとめは、XRT の PSF を考慮して決めなければならない。計算式 (3.2.1参照) から明らかなように、リージョンマップの刻みは、ARF の数値積分における PSF の値のサンプリングに対応する。したがって、リージョンマップをあまり大きくビンまとめしてしまうと、PSF の数値積分に支障をきたす。具体的には、XRT の PSF の空間変化のスケールは、大雑把に言って分角程度なので、10 ビン程度 (およそ 1/4 分角相当) にまとめることは問題ないであろうと思われる。

また、ポジションマップについては、天球上での輝度分布のスケールを考慮して決めなければならない。jbldarf はポジションマップで指定した通りの輝度分布について ARF を計算するので、例えば、輝度分布の起伏の空間スケールが 1 分角程度であるとわかっているものに対して、ポジションマップを 10 ビン (およそ 1/4 分角相当) 以上にまとめると、有効面積を不当に評価するおそれがある。

以上をまとめると以下のようになる。

(1) リージョンマップのビンまとめは、ビンまとめ後の 1 ビンの大きさが XRT の PSF の空間スケールよりも十分に小さくなるようにする。

---

[28] こうなると、スペクトル解析だけでなく、イメージ解析もからんでくるが…

(2) ポジションマップのビンまとめは、ビンまとめ後の1ビンの大きさがソースの輝度分布の空間スケールよりも十分に小さくなるようにする。

ただし、いずれの場合も、ビンまとめ数の決定は解析の主旨と密接に関わる問題である。解析者は、自分が行なっている解析の目的や必要な計算精度を十分に考慮し、その目的や精度にあったビンまとめ数を選ぶ必要がある。

3.5 応用編

ちょっと高度な使い方になるが、工夫すれば以下のような場合にも、jbldarf を適用することは原理的に可能である。しかし、§3.4 で述べた制限に引っかかる場合が多く、意味のあるものができるかどうかは、解析者がよく考えておく必要がある。

3.5.1 衛星の姿勢の揺れの補正

衛星の姿勢の揺れを補正した ARF を作る場合、その ARF を使うスペクトルが、検出器座標 (DET_X/Y) で集積したものか、天空座標 (X/Y) で集積したものかで厳密にはやり方が異なる。ここでは、それぞれについて考察してみる。

ただし、通常、衛星の揺れはせいぜい 1arcmin 程度なので、よほどリージョンを狭くとったり、GIS の井桁のそばのソースを扱ったりしない限り、姿勢の揺れがそれほど影響するとは思えない。また、$\sim 1arcmin$ 以上に姿勢が外れている部分のデータは、使わないようにするのが賢明であろう。さらに、現状では姿勢ファイルの精度が揺れに追従できていないので、これを行なっても虚しい場合が多い。

(1) 検出器座標で集積したスペクトルに対する ARF

姿勢が揺れるということは、ソース位置が時間によって動くということである。そのソース位置を $r(t)$ とした場合、$w_{\rm P}(\vec{r}_1)$ は次のように書ける。

$$w_{\rm P}(\vec{r}_1) = \int \delta(\vec{r}_1 - r(t))dt$$

従って、Position-type を 'FITS-file' か 'FFT-FITS-file' として Position-file に上記のものを入れておけば、姿勢の揺れが考慮されることになる[29]。

(2) 天空座標で集積したスペクトルに対する ARF

この場合は、ソース位置にしたがって、リージョンも動くことになる。だったら Position-file と同じく Region-file も姿勢の揺れでなまされたものを jbldarf に与えればいい[30]、と思うかもしれないが、ソース位置とリージョンは相関を持つので、その方法では正しい ARF は作れない。正しくは、ソース位置ごとにいちいちリージョンを決めて ARF を作り[31]、それをそのソース位置の重みをつけて足し合わせるという方法をとらなければならない[32]。式で書くと、(1) の $w_{\rm P}(\vec{r}_1)$ とそこでの ARF ($\mathrm{A}^{\vec{r}_1}[E]$ と書く) を使って、

$$\mathrm{A}[E] = \sum_{\vec{r}_1} w_{\rm P}(\vec{r}_1)\mathrm{A}^{\vec{r}_1}[E]$$

となる。jbldarf を何度も走らせる必要があるので、計算時間は (1) よりも多くかかる。

---

[29] このような Position-file をつくる機能は、私が知る限りでは XSELECT にはついていない。giphs を使う場合には makposmap というプログラムで作ることができる。

[30] この場合 Region-file は 0 か 1 だけではなく、そのピクセルの Exposure time に従って 0.5 とかの半端な値を持つことになる。

[31] 厳密には、回転の自由度もあるのでソース位置だけでは決まらないが、リージョンを円形にとった場合は、対称性により無視できる。それ以外の場合でも、リージョンが回転方向に 1arcmin 程度動いてもほとんど無視できるので、ソース位置だけで決まると思って問題ない。

[32] giphs を使う場合は、これを行なわせる shell script を発生する skyarf というプログラムがある。

3.5.2 広がったソースのレスポンス

ソースが広がっている場合でも、その全領域でスペクトルが同じだと仮定するなら、 Position-type を 'FITS-file' あるいは 'FFT-FITS-file' として、 Position-file にもともとのソースの形を入れることで、広がったソース用の ARF を作ることができる。本当は、もともとのソースの形はわからないはずであるが、他の衛星で得られているイメージやモデル、あるいは「あすか」のイメージを deconvolve したものを突っ込むのが妥当であろうと考えられる。こうやって作った ARF でスペクトルフィットした時のフラックスは、モデルで与えた全領域から出てくるフラックスとなる。しかし、大きく広がったイメージの場合、 XRT の PSF に「ちょうちょ」が入っていないことや、 PSF の tail がきちんと入っていないことが効いてくる可能性がある。

また、 GIS の場合、計算時間を考えると Position-type は 'FFT-FITS-file' にせざるをえないと思われるが、これだと PSF の場所による違いを反映することができないのも問題である。これを解決する策としては、 jbldarf の外で Ray-tracing を走らせてその結果を jbldarf に取り込む、という方法がある。これについては、次の節で紹介する。

3.5.3 Ray-tracing の結果の利用 (GIS のみ)

通常 jbldarf は、 XRT のレスポンス関数 (§3.4.1参照) を使って XRT の有効面積を評価し、 ARF レスポンスを計算する。それとは別に、レスポンス関数を使わずに XRT のモンテカルロシミュレーション (Ray-tracing) の結果を参照して ARF を計算する機能もついている[33]。

その際に必要なものは、各エネルギーバンドごとの、検出器面上での光子数分布[34]を表わす FITS のイメージファイルである。サイズは 256×256、光軸を中心 (128.5, 128.5) とし、座標軸は衛星座標の X/Y 軸と平行[35]、 1pixel のサイズは 0.25mm でなければならない。また、あるエネルギーに対する FITS イメージファイルの各ピクセルの値の総和は、そのエネルギーでの XRT の有効面積になるように規格化しておく必要がある[36]。

このようなイメージを、例えば エネルギー 1keV でのものは psf010.fits、 1.1keV でのものは psf011.fits、というような名前[37]でひとつのディレクトリ (仮に *directory* とする) に置いておく。そうして、jbldarf で Position-type に 'XRT-PSF' を、 Position-file に *directory*/psf%03d.fits を指定すれば[38]、jbldarf は ARF を計算する時にこれらの FITS ファイルにアクセスして XRT の有効面積を評価するようになる。より正確に言えば $psf(E;\vec{r}_2)$ (§3.3.2参照) が FITS ファイルから読み込まれる。 ARF 計算時に、計算すべきエネルギー E に対応するイメージが存在しない場合は (ほとんどの場合そうだと思うが)、その前後のエネルギーのイメージから直線補間した値が使われる。

この方法は、広がったソースの解析等、jbldarf に入っている XRT のレスポンスでは ARF レスポンスの十分な評価ができない時に有効である[39]。また、 Ray-tracing を用いてイメージファイルを作るのに必要な時間を度外視すれば、jbldarf の計算時間も一般に速くなる。しかし、現状では Ray-tracing のプログラムは XRT チーム以外の人が簡単に使える形にはなっていないし、仮に使えたとしても、得られた結果にどの程度のエラーがあるかはなんともいえない。

この機能の別の使い方として、「ちょうちょ」の形を SIS のイメージから得る、という方法も考えられよう。この場合、 Ray-tracing は用いずに SIS で観測された実際のイメージを光子数分布だと仮定する。イメージファイルの規格化に用いる有効面積は、例えば、jbldarf に入っている XRT レスポンスを使用する[40]。この場合 Ray-tracing にかかる時間がないため、短時間で ARF を作成できるという利点があるが、 SIS は視野が狭い、十分な統計があるとは限らない、などの理由でこの方法が使える場面も限定されるであろう。

---

[33] この機能は GIS に対してのみ有効である。

[34] もちろん、これを Ray-tracing で求める。

[35] 従って、検出器座標とは Y 軸の向きが反対である

[36] XRT の有効面積は、通常、半径 6 分以内を積分した時の値で定義されるので、「(点源の場合) 半径 6 分以内を積分した値が有効面積になるように規格化すべき」というのがより正確な言い方である。

[37] これを読んでお気づきだと思うが、イメージは 0.1keV 刻みでしか与えられない。

[38] このように指定すると、jbldarf は内部で sprintf() を用いて %03d のところに (エネルギー ×10) を展開する。エネルギーの単位は keV 。

[39] 実用例として、現在、 X 線源が全天一様に広がっていると仮定した CXB 用の ARF がこの方法で作られている。

[40] ARF ファイルの EFFAREA カラムを見れば簡単にわかる。

3.5.4　イメージ解析への利用 (GIS のみ)

jbldarf 本来の用途とは少し外れるが、DEBUG パラメータを ON にした時に吐き出されるファイルをイメージ解析に利用することができる。jbldarf はもともとのイメージ (Position-file) が XRT や検出器でなまされる効果を内部で計算しているので、./map/expo*nnn*.fits をエネルギーごとに重みをつけて足し合わせたものは、GIS で検出されたイメージと同じになっているはずである。これを逆手にとれば、いろいろな広がりのモデルを Position-file に入れて検出が予想されるイメージを作り、GIS のイメージと一番良く合ったものがもっともらしい広がりのモデル、といえよう。

もっとも、現状の XRT のレスポンスを考えると jbldarf が作ったイメージがどの程度正確に検出器上でのイメージを再現するかは疑問が残る。

第 4 章

関連ツール

この章で説明するツール類は jbldarf リリースキットの ./src/tools/ ディレクトリの中に収められている。

4.1 addarf

addarf は、複数の ARF ファイルの加重平均をとるプログラムである。名前には、'add' とついているが、実際は平均をとっているので気をつけて欲しい。使い方は、次のようになっている。

`addarf [-norm]` *Output-file* g_1 ARF_1 $\cdots$ g_n ARF_n

`-norm` オプションをつけた時とつけない時では、計算式は違っている。計算式は、出来る ARF を $\mathrm{A}[E]$、足し合わせる ARF を $\mathrm{A}_n[E]$、スケーリング係数を g_n とすると、次のようになる。

(1) `-norm` オプションをつけない時

$$\mathrm{A}[E] = \sum_{i=1}^{n} g_i \mathrm{A}_i[E] \Big/ \sum_{i=1}^{n} g_i$$

(2) `-norm` オプションをつけた時

$$\mathrm{A}[E] = \sum_{i=1}^{n} g_i' \mathrm{A}_i[E] \Big/ \sum_{i=1}^{n} g_i'$$

ただし、$g_i' = g_i \sum_E \mathrm{A}_{\mathit{1}}[E] \Big/ \sum_E \mathrm{A}_i[E]$

使い方としては、次の *SPEC1*, *SPEC2* のような 2 つのスペクトルとそれ用の ARF があった場合、$(\mathit{SPEC1} + \mathit{SPEC2})$ 用の ARF (ARF_{1+2}) は、近似的に次のようにして作ることができる。

スペクトル	積分時間	カウント数	ARF
SPEC1	T_1	N_1	ARF_1
SPEC2	T_2	N_2	ARF_2
$(\mathit{SPEC1} + \mathit{SPEC2})$	$T_1 + T_2$	$N_1 + N_2$	ARF_{1+2}

(1) `-norm` オプションを使わない場合

`addarf` ARF_{1+2} T_1 ARF_1 T_2 ARF_2

(2) `-norm` オプションを使った場合

`addarf -norm` ARF_{1+2} N_1 ARF_1 N_2 ARF_2

本来は別々にスペクトルフィットを行なうのが正しいのだろうが、足してしまった方が簡単に統計も上げられるし、XSPEC の画面も繁雑にならないですむ。

4.2 makemap

makemap は Region-file や Position-file を作るためのサンプルプログラムである。使い方は、次のようになっている。

`makemap` *teldef-file* *center-X* *center-Y* *outer-radius* [*inner-radius*] > *Output-file* jbldarf と同じく、座標は検出器ミリメートル座標で指定する。正しいマップを作るためには、jbldarf が使うのと同じ teldef ファイルを指定してやる必要がある。*inner-radius* を省略した場合は、0 がセットされる。リング状の領域しか指定できないため、これを単独で使用することにあまり意味はない[1]。しかし、こうやって作ったイメージを SIS のチップマップと AND をとったり (2.7.2 参照)、スムージングをかけたりすれば、使い道は広がるだろう。

4.3 mkarf.sh ◁U▷

mkarf.sh は shell script から jbldarf を使うサンプルプログラムであるが、座標の指定を検出器座標で行なえるという便利な特徴を持っている。使い方は、次のようになっている。

`mkarf.sh` *Output-file Detector-ID center-Xch center-Ych outer-radius*

上記のように引数を指定して動かしてやると、自動的に jbldarf を呼んで ARF を作成し、終了するようになっている。ただし参照する RMF は、shell script の中でセンサーごとに定義しているので、これをきちんと設定しておく必要がある。

なお、FTOOLS のバージョンによっては、**jbldarf.par** の中の パラメータの名前をすべて小文字にしないと、正常に動作しない場合がある。◁N▷

4.4 dummyrmf ◁U▷

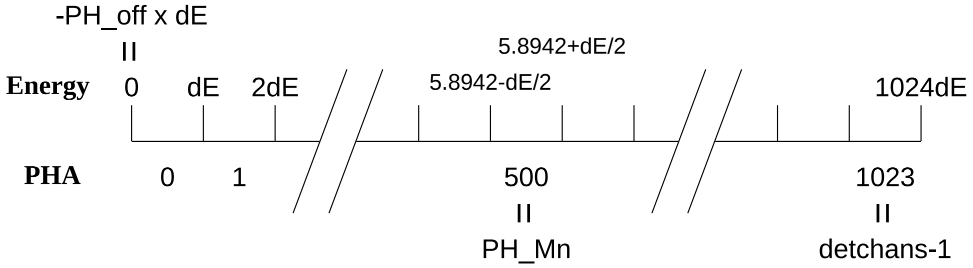
dummyrmf は対角成分のみが 1 の、ダミーのレスポンスファイルを作るプログラムである。使い方は、次のようになっている。

`dummyrmf` *Output-RMF-file PH_Mn PH_off Emin Emax detchans*

XSPEC で検出器のレスポンスを無視してスペクトルをフィットしたい時などに使えるだろう。FTOOLS のパラメータインターフェイスを使ったプログラムになっているので、実行には dummyrmf.par が必要である。各パラメータの意味は、次の通り。

Output-RMF-file	作成するダミーレスポンスのファイル名。
PH_Mn	5.8942 keV (Mn の K_α 線) のエネルギーが対応する PHA チャンネル。
PH_off	0 keV のエネルギーが対応する PHA チャンネル。
Emin	E_MIN がこれよりも大きいカラムが RMF に書かれる。
Emax	E_MAX がこれよりも小さいカラムが RMF に書かれる。
detchans	PHA のチャンネル数。

dummyrmf で作成した RMF レスポンスの Energy と PHA の関係は、$PH\_Mn = 500$、$PH\_off = 0$、$detchans = 1024$ とした場合、次の図のようになる。Energy 0 keV に対応するところが PHA の最初のビンの下端になること、PHA が 0 から始まること、の 2 点に注意して欲しい。



---

[1] ARF のヘッダに情報が残る分だけ、jbldarf に直接パラメータで指定した方が得である。

4.5 pressrmf

pressrmf は、RMF のマトリックス成分が 0 のところを圧縮して、RMF ファイルを小さくするプログラムである。使い方は、次のようになっている。

`pressrmf` *Input-RMF-file* [*Output-file*]

Output-file の指定がないときは、*Input-RMF-file* にオーバーライトされる。sisbldrsp2 で作られるレスポンスや、GIS チームからリリースされる RMF はすでに圧縮されているので、pressrmf をかける必要はない。

4.6 addarfrmf ◁N▷

addarfrmf は、複数の ARF×RMF ファイルの加重平均をとるプログラムである。平均のアルゴリズムは addarf と同様なので、そちらを参照されたい。使い方は次のようになっている。

`addarfrmf [-norm]` *Output-file* g_1 ARF_1 RMF_1 $\cdots$ g_n ARF_n RMF_n

addarf と異なって、作成されるファイルは RMF レスポンスであり、ARF も含んだものとなる。SIS の場合は、チップごとに RMF レスポンスも異なるので、addarf では不十分で、addarfrmf を使った方が良いだろう。

4.7 gisnortarf ◁N▷

jbldarf で作成した GIS の ARF レスポンスは、強力ライズタイムカット (gisclean) のかかったスペクトル用であるが、gisnortarf は、これを強力ライズタイムカットのかかっていないスペクトル用の ARF レスポンスへ変換するツールである。使い方は次のようになっている。

`gisnortarf` *ARF-file* *qdp-flag*

指定した *ARF-file* を直接書き換えるようになっている。*qdp-flag* は、変換定数を qdp ファイルに書きだすかどうかのフラグである。qdp ファイルの名前は、*ARF-file* で指定したファイルの拡張子を`.qdp` としたものが付けられる。ヘッダに gisnortarf をかけたという情報を書き込むので、2 回以上、同じ *ARF-file* に gisnortarf をかけることはできない。FTOOLS のパラメータインターフェイスを使ったプログラムになっているので、実行には gisnortarf.par が必要である。

第 5 章

FAQ

FAQ とは、「Frequently Asded Question の略で『よくある質問と、その回答集』と言う意味」です。(UNIX-SIG FAQ List with partial answers (Last Update: 19-Feb-1994) より)

質問 1. 誰が jbldarf を作っているの？

答. おもに、著者（石崎、平山）がプログラムを書いています。内訳を言うと、SIS ARF の計算部を平山、GIS ARF の計算部を石崎が書き、ユーザインターフェース部は石崎、平山が共同で書いています。XRT と GIS のレスポンスの部分はそれぞれ、XRT、GIS 両チームから公表された関数を使いますので、XRT、GIS 両チームの方々にもご参加いただいていることになります。

質問 2. どうして RMF を読むの？

答. jbldarf は、エネルギーの刻み (ENERG_LO、ENERG_HI のカラム) を RMF から読み込んで、同じ刻みで ARF を計算します。また、TELESCOP や INSTRUME などのヘッダの値も RMF からコピーされるようになっています。これらはいずれも、XSPEC でのスペクトルフィットの際のデータの整合性を確保するために行なっています。

なお XSPEC は RMF と ARF でエネルギーの刻みが同じになっているかどうかをチェックしないようなので[1]、RMF と ARF の組み合わせを間違えるととんでもないフィット結果が得られたりしますので、気をつけましょう。

質問 3. むかし作った ARF が XSPEC や addarf で読めなくなっちゃったんだけど

答. 0.50 より前の jbldarf では 1st extension の名前が 'ARF' となっていて、XSPEC 8.40 や jbldarf-0.50 に付属の addarf ではこのような ARF は受け付けられません。FTOOLS の fmodhead などを使って、次のようにヘッダを書き換えれば、読めるようになります。

```
EXTNAME = 'SPECRESP' / name of this binary table extension
```

質問 4. XSELECT で saoimage の中から選んだ領域をそのまま jbldarf に指定すると、大きさが微妙に違うような気がするのですが？

答. 一般に、XSELECT から選んだ方が領域が大きくなるようです。今のところはっきりとした原因はわかっていませんが、池辺さんが次のようなことじゃないかと教えてくれました。

IRAF では、ピクセルがちょっとでも指定した範囲に入っていれば、そのピクセルは選ばれたことになるそうです。一方、jbldarf では、ピクセルの中心が、境界を含む範囲に入っていれば選ばれたことになります。そのため IRAF で選んだ領域の方が普通大きくなるそうです。XSELECT も、これと同じように領域が指定されるのでしょう。

なお、互換性が損なわれるため jbldarf を上記のように変更する予定はありません。

---

[1] 少なくとも Version 8.34 ではそうだった。ビン数が同じかどうかはチェックする。

質問 5. jbldarf は FTOOL なの？

答. FITS ファイルを取り扱うツールではありますが、GSFC から出されている FTOOL の規定には沿っていないので、そういう点では FTOOL ではありません。ちなみに、GSFC からは、ARF 計算用の FTOOL として ASCAARF というものがリリースされています。詳しくは、fhelp などで調べてみて下さい。

質問 6. jbldarf は表街道なの？

答. 座標系などは表街道で使われているものを採用しています。そのため、場合によっては、裏街道ユーザには使いにくい面もあるかも知れません。しかし、GSFC からリリースされている訳ではありませんから、表街道とも呼べないでしょう。

質問 7. システムパラメータファイルがないっていわれるんですけど

答. このエラーメッセージは、環境変数の `SYSPFILES` で指定されるディレクトリに **jbldarf.par** がない時に表示されます。(FTOOLS 3.0 以降の libhost.a を使用した場合にこの問題が起こります)

解決方法としては、次の 3 つのやり方があります。

(1) システムマネージャに頼んで、**jbldarf.par** をシステム領域にコピーしてもらう。

(2) 環境変数の `PFILES` からシステムパラメータファイルのパスを除く。

(3) 環境変数の `PFCLOBBER` を unsetenv する。

方法 (1) がもっとも望ましいですが、やむを得ない時は、方法 (2) や (3) をとる必要があります。ただし、方法 (2) または (3) をとった時は、他の FTOOLS の動作にも影響を与えるので、注意して下さい。

Guide to ASCA_ ANL v2.00

by ASCA_ANL Working Group

H. Kubo[1], M. Hirayama[1], Y. Ishisaki[3], K. Mitsuda[2], M. Tashiro[1]
and
T.Takahashi[2]

[1]*Dept. of Physics, Univ. of Tokyo, Bunkyo-ku, Tokyo 113,JAPAN*
[2]*Inst. Space Astronautical Science, Sagamihara, Kanagawa 229,JAPAN*
[3]*Dept. of physics, Tokyo Metropolitan University, Hachioji, Tokyo 192-03, JAPAN*

March 5, 1997

Mailing List

The ASCA_ANL working group provides you a mailing list "`asca-anl`". With this mailing list you are announced a release of the latest version of the ASCA_ANL and you can get many useful information on ASCA_ANL. Because we announce all issues on the ASCA_ANL with the mailing list, those who are going to use the ASCA_ANL are strongly recommended to subscribe to the mailing list.

The mailing list is also aimed to exchange information on the ASCA_ANL between users and the working group members. Please inform us your requests or bugs you found. Any questions on the ASCA_ANL are welcomed.

To post your issues to the mailing list, send a mail containing your issues to

```
asca-anl@phys.s.u-tokyo.ac.jp
```

To subscribe to the mailing list, send a mail containing only

```
SUBSCRIBE
```

to

```
asca-anl-request@phys.s.u-tokyo.ac.jp
```

Note that the address

```
asca-anl-request@phys.s.u-tokyo.ac.jp
```

is for an automatic subscription system and DO NOT send your request on the ASCA_ANL to this address.

ASCA_ANL_contrib

The ASCA_ANL working group offers an opportunity to exchange useful modules created by the ASCA_ANL users. The ASCA_ANL_contrib is a system to gather useful modules and useful functions as user's contribution and to re-distribute them to users.

There are two ways to access the ASCA_ANL_contrib.

1. Anonymous FTP
   By courtesy of R.Fujimoto and other people at ISAS, we can use a following anonymous FTP servise for the ASCA_ANL_contrib.

   `ftp://ftp.astro.isas.ac.jp/asca/software/isas/asca_anl_contrib/`

   This direcotory can also be seen as

   `ISAS:/xray/s1/ftp/software/isas/asca_anl_contrib/`.

   Read `README` (in Japanese) first.

2. WWW (World Wide Web)
   We also put the same information as a hyper text on WWW. You can browse the same information with "Mosaic".

   `file://ftp.astro.isas.ac.jp/asca/software/isas/asca_anl_contrib/Summary.html`

   Read `About ASCA_ANL_contrib` (in Japanese) first.

Any contribution on the ASCA_ANL of you are welcomed. Your contribution is not necessarily an ASCA_ANL module. Usefule function, useful shell script, useful program, all these are accepted as your contribution as far as it is related to the ASCA_ANL. There are some restrictions to put your contribution to the ASCA_ANL_contrib. It is described in `README` in the anonymous ftp directory and in `About the ASCA_ANL_contrib` on WWW. Please read it carefully before you put your contribution.

We are looking forward to your contributions.

Contents

List of Figures

Chapter 1

Introduction

The ASCA_ANL is an analysis frame work [1] for the ASCA experiment specially designed for those who may write programs for their own analysis in "human" way. It accepts a set of user-supplied[2] simple modules and handles the execution sequence of the modules. The sequence can be dynamically changed and a user can specify the modules to be executed and the order of execution.

The ASCA_ANL provides standard environment for the programmer. It has several built-in modules to access photon information in the science FITS file together withe the orbital information in the orbit FRF and the HK FITS file[3]. This feature enables a user to analyze ASCA data in a "human" way. It reads data from the FITS file produced by the FRFread/ASCALIN/XSELECT and pass them to the user's module. Thus the user does not have to know details of FITS I/O nor a format of the orbit FRF. Furthermore, he/she is allowed to perform his/her original analysis because the module can be written easily as the user pleases, even if it is not supported in XSELECT.

By standardizing the way to write programs, the user is benefited in several ways from the analysis point of view. First, since most part of user interface is provided by the ASCA_ANL system, all software written in the ASCA_ANL looks alike. The same look-and-feel is saves your time to learn how to use a new program every time. Second, the user can easily combine modules written by different people. Numerous ways of analysis can be realized simply by combining "ready-made" modules.[4] Third, Once a user module is established in the ASCA_ANL, the module can be easily transformed to a standard FTOOL package[5] as expected by people living in the "human" world.

The ASCA_ANL uses various software packages in high energy experiments developed so far. Especially, HBOOK4[6] is a standard histogram package in the system that allows the user to make as many histograms (spectra) as he/she desires. DISPLAY45[7] reads histogram files stored by HBOOK4 in the ASCA_ANL and allows the user to operate histograms in various ways. These tools are useful and powerful as well as they are well-maintained. In addition, it converts a format of the histogram into a spectral FITS format and saves it in a spectral FITS file for XSPEC.

In this documentation, we introduce a concept of ANL system. We present a "walk-through"

---

[1] The ASCA_ANL is "humanistic" software rather than "animal" software. The key of the ANL system is the "software-pipeline". The concept of the ANL system was first invented by the PEP4 group at LBL and the scheme was used in the analysis extensively.

[2] In the ASCA_ANL world a "user" means a person who write programs for their analysis. Thus, you can say a user is a programmer and analyzer.

[3] the BNK/EVS system is used to handle the communication between modules.

[4] For example, the ASCA_ANL has a built-in module that reads FRF file directly. Since the system manages to deliver the photon data is the same way as the one for FITS data, the same module can analyze data both from FITS files and from FRF files.

[5] By just adding standard I/O routines required for the FTOOL package.

[6] A standard histogram package in CERN Library.

[7] DISPLAY45 is a command-driven interactive histogram browsing program

guide, instructions to access data, and a template of a module. Also, an example of user modules are listed in appendix.

Chapter 2

Getting Started

The best way to learn an analysis system is to use it. In this chapter[1] we describe the way to analyze ASCA data using the ASCA_ANL system with a sample program `gisspec` (see below) as an example. We recommend you to read this chapter in front of your computer following our directions below.

To clarify the directions, let us assume:

1. the ASCA_ANL has already installed in your computer
2. the ASCA_ANL is installed in a directory `/asca_anl/`
3. You run the ASCA_ANL on a directory `~/work/`
4. *Display45* has already installed in your computer

2.1 Compile Sample Program

Some examples of a user program are attached to the distribution kit of the ASCA_ANL. Each module in the samples is written both in FORTRAN and in C. We hope they help you to understand the ASCA_ANL system[2].

You can compile the samples easily as follows:

1. Copy source files of a sample you want to compile.
2. Modify include files for your computer system (if needed).
3. Make the sample.

2.1.1 Copy a Sample

Each directory below a directory `sample` corresponds to a sample program. The name of each directory is self-explanatory. Find a program you want to compile among them and copy the directory to your local directory.

In our case, type

```
% cd ~/work
% cp -r /asca_anl/sample/gisspec .
```

---

[1] You can skip this chapter if you have an experiment to analyze data with ASCA_ANL.

[2] the samples are written carefully from "ASCA_ANL" point of view. Each sample is full of recommendations by the ASCA_ANL working group. We hope you master how to write your module by referring the samples.

2.1.2 Arrange Makefile

Before compiling the sample, you have to modify[3] a file `Includes.make` to fit you computer sysytem.

```
% cd gisspec
% emacs Includes.make
```

In the file Includes.make several macros are defined to specify the directory structure of your computer system. Edit the file and change definitions of macros appropriately.

Each macro specifies the directory which contains a library or header files for the library and how to compile and link them to the ASCA_ANL system. Table2.1 shows required contents in the directory specified by the macros. Also, option flags for a compiler and a linker are also listed in Table2.1.

`MACRO`	Directory Contents etc.
`ASCA_ANL_DIR`	the ASCA_ANL directory
`ASCA_ANL_INC`	header files of the ASCA_ANL
`ASCA_ANL_LIB`	libraries of the ASCA_ANL
`ASCA_ANL_BIN`	executables for the ASCA_ANL
`ASCA_ANL_LNK`	link option for the ASCA_ANL
`ASCATOOL_DIR`	ascatool directory
`ASCATOOL_INC`	header files for ascatool `ascatool.h`, etc.
`ASCATOOL_LNK`	link option for ascatool `libascatool.a`
`COM_CLI_LNK`	link option for CLI/COM `libCOM.a`, `libCLI.a`, `libreadline.a`
`ATFUNCTIONS_INC`	header files for atFunctions `atFunctions.h` and `atError.h`
`ATFUNCTIONS_LNK`	link option for the atFunctions `libatFunctions.a`
`CERN_DIR`	CERN directory
`CERN_INC`	header files for CERN library `cfortran.h` and `hbook.h`
`CERN_LNK`	link option for CERN library `libgraflib.a`, `libgrafX11.a`, `libpacklib.a`, `libkernlib.a`, `libmathlib.a`, `libX11.a`
`FITSIO_LNK`	link option for FITSIO `libfitsio.a`
`CC`	default C compiler
`FC`	default FORTRAN compiler
`DEFAULT_F77_LNK`	default FORTRAN library

Table 2.1: Macros in `Includes.make`

2.1.3 Make

In order to "make" it, just type

---

[3] In most cases you can skip this proceedure because `Includes.make` is modified so by a person who installed the ASCA_ANL.

```
% make
```

in the current directory. When all compilations are done, executables `gisspec_f` and `gisspec_c` are created in the current directory. `gisspec_f` corresponds to `gisspec_f.f` and `circreg_f.f`, source files of modules `GISSPEC` and `CIRCREG` written in FORTRAN and `gisspec_c` is created from `gisspec_c.c` and `circreg_c.c`, source files of modules `GISSPEC` and `CIRCREG` written in C. A module `FITSREAD` and A main program of the ASCA_ANL were also linked to the resultant executables.

To run `gisspec_f`, type

```
% gisspec_f
```

and to run `gisspec_c`, type

```
% gisspec_c.
```

Very few differences[4] are resulted from two executables.

2.2 Walk Through the ASCA_ANL

Now let us see an example of analysis with the ASCA_ANL. Since the usage of any user program are similar in its user interface, you can analyze data with this program by following the instructions in this section.

We compiled a program `gisspec_f` and `gisspec_c` both of which includes “FITSREAD”, “CIRCREG”, and “GISSPEC” as linked modules. By typing `gisspec_f` or `gisspec_c` in a command line you can see a command menu of the ASCA_ANL and the prompt ”ANL>”.

```
%gisspec_f

 ANL:  Program ASCA_ANL *** ASCA Analysis ***

 DEFINE_ANALYSIS SHOW_ANALYSIS   MODIFY_PARAM
 BOOK_HISTOGRAM  ANALYZE_DATA    QUIT

 ANL: Select  1 Option
ANL>
```

The first thing you have to do is to define an analysis chain. To define the analysis chain, type ”DEFINE_ANALYSIS” or its abbreviation. The command name can always be abbreviated and the ASCA_ANL does not distinguish between upper case and lower case.

```
ANL> define

 ANL:  Program ASCA_ANL *** Define Analysis ***

 NONE     FITSREAD CIRCREG  GISSPEC  FULL

 ANL: Select  1 or More, Then OK
ANL>
```

In this menu you can see all the modules you linked in `gisspec_f`. Also, names `NONE` and `FULL` are displayed. These two are reserved in ASCA_ANL and explained below.

Suppose that we read ASCA data from a science FITS file, select events in a circular region on GIS images, and create GIS spectra and GIS images. To do these we need three modules; `FITSREAD` to read data from a FITS file, `CIRCREG` to select events in the specified circle on GIS images, and `GISSPEC` to obtain GIS spectra and GIS images. You can choose these modules in this order by typing `FITSREAD CIRCREG GISSPEC ok`. The ASCA_ANL returns to the command menu by accepting “ok”.

---

[4] Possible differences are the number of spaces or blank lines in output or something minor.

```
ANL> FITSREAD CIRCREG GISSPEC ok

 ANL:  Program ASCA_ANL *** ASCA Analysis ***

 DEFINE_ANALYSIS SHOW_ANALYSIS   MODIFY_PARAM
 BOOK_HISTOGRAM  ANALYZE_DATA    QUIT

 ANL: Select  1 Option
ANL>
```

If you type `FULL ok` instead of `FITSREAD CIRCREG GISSPEC ok`, all the modules displayed in the menu are chosen, *i.e.*, the effect of typing `FULL ok` is, in this case, completely the same as you typed. If you type `NONE ok` instead, none of the modules is chosen.

Now let us return to our analysis.

After defining the analysis chain, you can modify parameters you use in our analysis. To do this, type `MODIFY_PARAM` in the command menu.

```
ANL> modify

 ANL:  *** Modify Parameter ***

 NONE     FITSREAD CIRCREG  GISSPEC  FULL

 ANL: Select  1 or More, Then OK
ANL>
```

In this menu you can see only three modules you selected in `DEFINE_ANALYSIS`. By typing `CIRCREG ok` you are asked to modify parameters for the module `CIRCREG`.

```
ANL> circreg ok

 *** Circular Region ***

 Sensor to set (-1=end of set): 0 ? 0
 DETX of Center (pixel): 640.5 ? 0.0
 DETY of Center (pixel): 640.5 ? 0.0
 Radius (pixel): 1024.0 ? 150.0

 *** Circular Region ***

 Sensor to set (-1=end of set): 0 ?
```

You can specify parameters as you desire by answering values as is asked. In above case we set a image region for SIS–0 centering (0.0, 0.0) with 150.0 pixels as its radius.

The values printed left to a question mark is a default value. If you can accept the default, you can omit inputing a value, *i.e.*, all you have to do is to hit a return key.

```
 *** Circular Region ***

Sensor to set (-1=end of set): 0 ? 1
DETX of Center (pixel): 640.5 ?
DETY of Center (pixel): 640.5 ?
Radius (pixel): 1024.0 ?

 *** Circular Region ***

 Sensor to set (-1=end of set): 1 ?
```

In above case we set a image region for SIS–1 centering (640.5, 640.5) with 1024.0 pixels as its radius. Note that we did NOT type coordinates of the region center BUT only hit a return key.

There is another way to specify parameters. Now you know the order of parameters to be asked, it is a little bit boring to hit the return key after typing a value. You can put values in one line like this:

```
 *** Circular Region ***

 Sensor to set (-1=end of set): 1 ? 2 0.0 0.0 16.0


 *** Circular Region ***

 Sensor to set (-1=end of set): 2 ?
```

In above case we set a image region for GIS–2 centering (0.0, 0.0) with 16.0 pixels as its radius.

At this point perhaps you feel uncomfortable by watching the input line because the input line is difficult to remember which value means what. Now you know the begining `2` means GIS-2, first `0.0` means `DETX` of the region center, and so on. But, you may forget its meaning one month after.

For such people there is yet another way of specifying parameters. You can put meanings of the values to some extent in case of setting integer or floating point paramters.

```
 *** Circular Region ***

 Sensor to set (-1=end of set): 2 ? g3 x=0.0 y=0.0 r=16.0


 *** Circular Region ***

 Sensor to set (-1=end of set): 0 ?
```

In above case we set a image region for GIS–3 centering (0.0, 0.0) with 16.0 pixels as its radius.

How about this!

In case of setting integer values the system[5] ignores a part of the input line which can not be recognized as an integer (`g` of `g3` in above case). The same is true in case of floating point values.

Note that no space is allowed between `g` and `3`. Also, `x=0.0` must not include any space between `x`, `=`, and `0.0`. The others follow the same restriction.

After setting parameters type `-1` to return to the command menu as is prompted. Then you will be back to the command menu again.

```
 *** Circular Region ***

 Sensor to set (-1=end of set): 3 ? -1

 ANL:  Program ASCA_ANL *** ASCA Analysis ***

 DEFINE_ANALYSIS SHOW_ANALYSIS   MODIFY_PARAM
 BOOK_HISTOGRAM  ANALYZE_DATA    QUIT

 ANL: Select  1 Option
ANL>
```

The next is to book histograms. Type a command `BOOK_HISTOGRAM` or its abbreviated form and Select modules to book histograms for.

In our case histograms should be booked for the module `GISSPEC`. So type `GISSPEC ok`.

```
ANL> book

 ANL:  *** Book Histogram ***

 NONE     FITSREAD CIRCREG  GISSPEC  FULL

 ANL: Select  1 or More, Then OK
ANL> gisspec ok

 ANL:  Program ASCA_ANL *** ASCA Analysis ***
```

---

[5] Exactly speaking, this is a feature of the CLI, a standard user interface routines used in the ASCA_ANL.

```
 DEFINE_ANALYSIS SHOW_ANALYSIS   MODIFY_PARAM
 BOOK_HISTOGRAM  ANALYZE_DATA    QUIT

 ANL: Select  1 Option
ANL>
```

You can confirm the analysis chain you defined now by commanding `SHOW_ANALYSIS`.

```
ANL> show

                   **************************
                   ***** Analysis chain *****
                   **************************

     [ 1] FITSREAD         ------
     [ 2] CIRCREG          ------
     [ 3] GISSPEC          BOOKED

 HBOOK_BUF_SIZE =      2200000
 BNK_BUF_SIZE   =       120000

 ANL:  Program ASCA_ANL *** ASCA Analysis ***

 DEFINE_ANALYSIS SHOW_ANALYSIS   MODIFY_PARAM
 BOOK_HISTOGRAM  ANALYZE_DATA    QUIT

 ANL: Select  1 Option
ANL>
```

The number in a bracket shows the order of procedures in our analysis. `BOOKED` at right most column means that histograms for the module have already been booked and `------` means not. `HBOOK_BUF_SIZE` and `BNK_BUF_SIZE` are sizes of buffers (in unit of four bytes) used for HBOOK and BNK, respectively.

Now you are ready to analyze ASCA data.

By commanding `ANALYZE_DATA` you can proceed next stage of ASCA_ANL. In this stage you can read data, analyze data, save histogram, clear histogram, and so on. On the way to this stage the `SHOW_ANALYSIS` command is executed automatically. Also, `PSEUDO_INTERVAL`[6] is printed out by `FITSREAD` in our case. Now you don't have to worry about `PSEUDO_INTERVAL` because it is set to zero as a default. That means nothing special happening to our analysis.

```
ANL> ana

                   **************************
                   ***** Analysis chain *****
                   **************************

     [ 1] FITSREAD         ------
     [ 2] CIRCREG          ------
     [ 3] GISSPEC          BOOKED

 HBOOK_BUF_SIZE =      2200000
 BNK_BUF_SIZE   =       120000

 ANL:  *** FITSREAD show parameter ***

     PSEUDO_INTERVAL    0.0000 (sec)

 ANL:  Program ASCA_ANL *** ASCA Analysis ***

 READ_DATA    SAVE_HIST    CLEAR_HIST   SHOW_STATUS
 RESET_STATUS EXIT

 ANL: Select  1 Option
ANL>
```

---

[6] See Section6.1 in details.

Command `READ_DATA` starts the analysis. You are asked to enter a name of a science FITS file you want to analyze.

```
ANL> read
Input fits filename: frf.fits ? ../testdata/dummysis.fits
 successfully opened
number of rows          128
Instrument : SIS0
Data Mode : BRIGHT
Bit Rate : HIGH
 FTISREAD: Can not get STDGTI (status=         107)
 FTISREAD: use TSTART and TSTOP keywords
```

After replying the file name `FITSREAD` opens the data file. Then you are asked the number of events you want to analyze. If you want analyze all data in the input FITS file, reply `-1`.

```
Number of events (-1=all, 0=exit): -1 ? -1
```

You are also asked a frequency of displaying the number of data analyzed until that time. If you specify 500, you will see the number of analyzed data displayed every 500 events. In our case total number of events (printed by `FITSREAD`) is only 128, let us specify `10` as a printout frequency.

```
Event number printout frequency: 100 ? 10
```

Just after answering the printout frequency the ASCA_ANL starts processing data.

```
 Event...           1
 Event...          11
 Event...          21
 Event...          31
 Event...          41
 Event...          51
 Event...          61
 Event...          71
 Event...          81
 Event...          91
 Event...         101
 Event...         111
 Event...         121

 ANL:  Program ASCA_ANL *** ASCA Analysis ***

 READ_DATA    SAVE_HIST    CLEAR_HIST   SHOW_STATUS
 RESET_STATUS EXIT

 ANL: Select  1 Option
ANL>
```

After finishing analyzing data in the specified file If you have another FITS file to be analyzed, type `READ_FILE` again and repeat the procedure after that.

With all the data analyzed, it is time to exit from this program. That is simple. Type `EXIT`. Then you will see the version numbers of modules in the analysis chain you defined and the statistics of the analysis you did before you exit.

```
ANL> exit

                  **************************
                  ***** Analysis chain *****
                  **************************

         ASCA_ANL         version 0.81
     [ 1] FITSREAD         version 0.73
     [ 2] CIRCREG          version 1.0
     [ 3] GISSPEC          version 4.0
```

```
 *** results of Event selection *** < Number of selects :   26 > (EVS vers.2.6)
          0 : SIS:SIS0 Event
          0 : SIS:SIS1 Event
        128 : GIS:GIS2 Event
          0 : GIS:GIS3 Event
          0 : SIS:PSEUDO Event
        128 : GIS:PSEUDO Event
          0 : SIS:SIS0 Pseudo
          0 : SIS:SIS1 Pseudo
        128 : GIS:GIS2 Pseudo
          0 : GIS:GIS3 Pseudo
          0 : ANL:BIT HIGH
          0 : ANL:BIT MEDIUM
          0 : ANL:BIT LOW
        256 : GIS:PH mode
          0 : GIS:MPC mode
          0 : SIS:BRIGHT mode
          0 : SIS:BRIGHT2 mode
          0 : SIS:FAINT mode
          0 : SIS:FAST mode
          1 : FITSREAD:BEGIN
        256 : FITSREAD:ENTRY
        256 : FITSREAD:OK
        128 : FITSREAD:PSEUDO
        128 : FITSREAD:OBS
          1 : CIRCREG:BEGIN
        256 : CIRCREG:ENTRY
        134 : CIRCREG:OK
          1 : GISSPEC:BEGIN
        134 : GISSPEC:ENTRY
        134 : GISSPEC:OK
```

The last question is if you need to save histograms or not. You can answer `Y` or `N`. By entering a file name for histograms to be saved you will exit from this program. If your specified file already exists, you are also asked if you want to delete the old one before you exit.

```
Save histogram (Y/N): Y ? Y
File Name: histfile.hbk ? test.hbk
'test.hbk' already exists, delete (Y/N): Y ? Y
```

Now you get the result of analysis in a file "test.hbk". You can operate these histograms very easily with *display45*. Here we just look at resultant histograms with *display45*.

2.3 Browse Results with Display45

We have a results from our own analysis of ASCA data. Let us look at the results with *display45*. Since *display45* displays histograms with opening X-window, you have to set an environment variable `DISPLAY`.

```
% setenv DISPLAY <your-terminal>
```

To invoke *display45*, type

```
% dis45 test.hbk
 Version 1.18/08 of HIGZ started
DIS> @%
%output off
%output on
DIS> FETC test.hbk 0 0
DIS> (End of File)
DIS>
```

The X-window is opened and you are prompted to enter a command for *display45*. At first let us see what kind of histograms are stored in `test.hbk`. Type `i` to do this.

```
DIS> i
       1   100 1  GIS2 PHA
       2   200 1  GIS3 PHA
       3   300 1  GIS2 PI
       4   400 1  GIS3 PI
       5   500 2  GIS2 RAWX vs RAWY
       6   600 2  GIS3 RAWX vs RAWY
DIS>
```

In the left most column sequencial numbers of the histograms are shown. In the second column you can see "histogram ID" to specify histograms. The third colunm means its dimension and the right most column shows its titles.

For example, the first line means:

- histogram ID is 100
- one dimensional histogram
- title is `GIS2 PHA`

To display the histogram with its ID of 100 type `h 100 t` which means "select histogram with ID 100, and type it."

```
DIS> h 100 t
   100 1  GIS2 PHA
DIS>
```

You can see a PHA spectrum of GIS–2, one of the result from our analysis. If you want to browse a next histogram (whose ID is 200), just type `nt` which stands for "select next histogram, and type it".

```
DIS> nt
   200 1  GIS3 PHA
DIS>
```

You are watching a PHA spectrum of GIS–3 now. You can also browse PI spectra and GIS images in the same way.

To exit from *display45* just type `exit`.

```
DIS> exit
Complete Display 45...
%
```

Display45 features poweful functions to operate histograms which I hope are helpful for your analysis. Also, it includes a fitting package called *MINUIT* which is well established. See a manual of *display45* distributed with *display45* for how to use these functions.

Chapter 3

On Your Own

After learning how to 'use' the ASCA_ANL it is time to know how to write your own analysis tools. Since the ASCA_ANL provides you good environment for ASCA analysis, it is very easy to write your analysis program. However, and you have to know rules on writing programs on the ASCA_ANL before you start writing. Also, a few tools are provided for you to create new program easily.

In this chapter we show you a way to create your original program on the ASCA_ANL. In some case the program `gisspec_f` (or `gisspec_c`) is quoted as an example. This program is attached to the distribution kit of the ASCA_ANL. and explained its usage in this manual (See §2.2). You are expected to be familiar to how to use this program. If not so, ask how to use the program or read Section2.2 before you begin to read here.

3.1 What is the Module

First of all, what is called the module in the ASCA_ANL? To create a new and your original module you had better know its structure and functions in the ASCA_ANL. In this section we will outline how to create the module for the ASCA_ANL. It includes an explanation of a structure of the module and some rules and recommendations on writing the module. Most part of this section is the same for FORTRAN and C, although detailed rules can be different between the languages as mentioned later in this manual.

3.1.1 Structure of the Module

The module of the ASCA_ANL consists of seven subroutines[1]. In order to create a new module named `USER`, for example, you need subroutines named

- `USER_init` (Initialization routine),
- `USER_com` (Communication routine),
- `USER_his` (Histogram booking routine),
- `USER_bgnrun` (Beginning-of-Run routine),
- `USER_ana` (Analysis routine),
- `USER_endrun` (End-of-Run routine), and

---

[1] A calling subroutine is written in FORTRAN. So, one can say FORTRAN is a "native" language of the ASCA_ANL. But, you also can write your module in C, of course, almost in the same fashion.

- `USER_exit` (Exit routine).

These routines are called by the main program of the ASCA_ANL. The subroutines are required even if you do not need their functions. For example, you have to prepare `USER_his` even if you do not have to book any histogram. In such a case you can write a subroutine which contains `Return` statement only.

Anyway, you don't have to understand details in each routine at present. But all you have to know here is "the module consists of seven basic routines".

3.1.2 Module in the ASCA_ANL

The seven subroutines in the module are called by the ASCA_ANL system. A relationship between each subroutine and a command in the command menu is be summarized as following:

`USER_init` corresponds to `ANALYZE_DATA`.
This subroutine is called while jumping into the second stage of analysis, *i.e.*, after entering the command `ANALYZE_DATA` and before new command menu (includes `READ_DATA`, etc.) showing up.

`USER_com` corresponds to `MODIFY_PARAM`.
This subroutine is called for only modules that is selected by `MODIFY_PARAM` command. Since this command can be invoked any times, you can call this subroutine any times.

`USER_his` corresponds to `BOOK_HISTOGRAM`.
This subroutine is called for only modules that is selected by `BOOK_HISTOGRAM`. Since it is illegal to book histogram more than once, this subroutine can be called once for one module.

`USER_bgnrun` corresponds to `READ_DATA`.
This subroutine is called once for one data set to be processed just after `READ_DATA` command is invoked.

`USER_ana` corresponds to `READ_DATA`.
This subroutine is called N times in processing N photons after a series of calls of "_bgnrun" entries.

`USER_endrun` corresponds to `READ_DATA`.
This subroutine is called once for one data set to be processed after a series of calls of "_ana" entries.

`USER_exit` corresponds to `EXIT`.
This subroutine is called just after `EXIT` command is invoked.

In addition, relationship between the commands and subroutines in a module are illustrated in Figure 4.1. Note that order of subroutine calls is always the same as defined by the command `DEFINE_ANALYSIS`. Also, some of subroutines above are called less times than described above in case that the analysis flow is controlled by the return values of the subroutines. See §4.2.1 for details on return values.

3.2 How to Create the Module

Now you are ready to create your own module to analyze ASCA data in your own way. As writing usual program on your computer system, you have to write a source file at first. But, not as in usual cases, there are many restrictions and recommendations in writing a module. In this section we show the way to write a module for your analysis while also summarizing such rules on a module from a programmer's points of view.

3.2.1 mkanlmodule

A useful command 'mkanlmodule' is also in the distribution kit. This command creates a template for the module. If you creates it with this command, most of recommendations are satisfied automatically. Also, you can choose in which language, FORTRAN or C, you want to write module.

You can invoke this command by setting an appropriate path to `PATH` environment variable. This command should be in the directory specified by `ASCA_ANL_BIN` in the file Includes.make[2] If you don't know where it is, please ask a person who installed the ASCA_ANL in your system where the `ASCA_ANL_BIN` means.

Before invoking `mkanlmodule` you have better know that it has two different modes to interface you, *i.e.*, an interactive mode and a command line mode. In an interactive mode you are prompted to answer a few questions on how to create the template of your module. In a command line mode you can specify all parameters in a command line. You can find below how to use `mkanlmodule` in both modes.

Interactive Mode

First, type

```
% mkanlmodule
```

without any command line option. You are asked for a name of the module you intend to write.

```
Enter module name:  ?
```

Now let us assume you are going to create the module named 'USER'. In that case, type `USER` against the prompt.

```
Enter module name:  ? USER
```

You are asked for 'your language' in turn.

```
Do you write it in FORTRAN (Y/N): Y ?
```

If it is FORTRAN, reply `Y`. Reply `N` in case of C language.

```
Do you write it in FORTRAN (Y/N): Y ? Y
```

In above case we choose FORTRAN language. The next and last question is for a name of an output file.

```
Output file: USER.f ?
```

In this example `USER.f` is a default name of an output file. If you accept this name `USER.f`, hit a return key. If not, specify the name like this:

```
Output file: USER.f ? user.f
 Empty module written in user.f
```

Now you can see a file `user.f` is created in a current directory. Look into the file. That is a template for the module, *i.e.*, an empty[3] module for the ASCA_ANL.

---

[2] We guess you already know this file because you have compiled the program `gisspec_f`.

[3] Can you believe it is 'empty'?

Command Line Mode

You can invoke `mkanlmodule` in 'command line mode' by invoking it with command line options. In this mode you have to specify all the parameter on what your module is like.

As in the previous section, for example, let us assume that you want to create the module named 'USER' in FORTRAN and that a name of an output file is 'user.f'. Then you can get the file by typing like this:

```
% mkanlmodule -FORTRAN -o user.f USER
```

or just typing

```
% mkanlmodule USER > user.f
```

Above two command generates exactly the same results.

Synopsis of this command is the following and command line options for `mkanlmodule` are listed in Table3.1.

```
% mkanlmodule [ option ] module_name
```

Option	Function
`-FORTRAN`	Forces to create a module written in FORTRAN (default).
`-Fortran`	the same as above.
`-C`	Forces to create a module written in C language.
`-o file`	Specifies an output file name. An outpur is redirected to a standard output when file is '-' (default).

Table 3.1: Command line option for `mkanlmodule`

3.2.2 Coding Analysis

Let us start coding your own analysis by filling an empty module with mkanlmodule. It is a good idea that we start to fill a subroutine `USER_ana`. The subroutine `USER_ana` is called photon by photon; that means here placed an essence of your original way of analysis. Hereafter, we explain how to fill `USER_ana` with reffering `gisspec_f.f` as an example.

In `USER_ana` you process each photon as following:

1. get photon information[4],
2. process the photon, and
3. put a new information on the photon(if needed[5])

In order to access information on each photon the `EVS` and `BNK` system are used. In `gisspec_f.f`, we intend to analyze only GIS PH mode data you can select them by calling a logical function `EVS` like this (you can see this at the beginning of `GISSPEC_ana`.):

```
c check if PH mode or not
      If ( EVS('GIS:PH mode') ) then
         Continue
      Else
         Write(*,*) 'DATAMODE is not PH'
         status = ASCA_ANL_QUIT
         Return
      Endif
```

---

[4] For example, photon arrival time, PHA of the photon, DETX of the photon, etc.

[5] This procedure is necessary in case you create a filter module.

An argument of the logical function `EVS` is called "EVS keyword". In above case `'GIS:PH mode'` is an EVS keyword[6] to know whether the photon is read out in GIS PH mode or not. If the current photon had arrived during GIS PH mode, `If` statement causes an execution of `Continue`. Otherwise, *i.e.*, in case of GIS data in other mode or SIS data,

```
            Write(*,*) 'DATAMODE is not PH'
            status = ASCA_ANL_QUIT
```

are executed and returned to a calling routine in the ASCA_ANL.

By returning with `ASCA_ANL_QUIT`[7] set to `status` (the third argument of `GISSPEC_ana`) you can force the ASCA_ANL discard the whole data set which is read by `READ_DATA` command(See §2.2). In above case only GIS PH mode data is selected and others are discarded. In the same way By returning with `ASCA_ANL_SKIP` set to `status` you can force the ASCA_ANL discard the current photon. This feature is useful for a filter module. See §4.2.1 for details.

After selecting data let us get parameters of the photon by calling `BNKget` subroutine. If you need to know a sensor ID, an energy of a photon, etc., call a subroutine `BNKget` like in `gisspec_f.f`.

```
c get sensor id
      CALL BNKget('ANL:SENSOR', sizeof(sensor), size, sensor)

c get pha and pi
      CALL BNKget('GIS:PHA', sizeof(pha), size, pha)
      CALL BNKget('GIS:PI', sizeof(pi), size, pi)
      .....
```

In this example `size`, `sensor`, `pha`, and `pi` are integer variables in `gisspec_f.f`. After the call, for example, `pha` is set to a value of "PHA column" in an analyzed science FITS file[8]. `size` is set to a size of data actually got by the call. The variable `size` is equal to four in an above case if the data is successfully obtained.

The first argument of the subroutine `BNKget` is called BNK keyword. `'GIS:PHA'` is a BNK keyword to obtain a value of "PHA column" in a FITS file. The BNK keyword `'GIS:PHA'` is reserved by the ASCA_ANL system. The reserved BNK keywords are listed in Table 4.6 in Appendix 6.1. The size of data you should specify as a second argument of `BNKget` are also listed in the table.

The last process in `GISSPEC_ana` is to fill data into histograms. You can assume the histograms are booked preceeding to `USER_ana` calls by `BOOK_HISTOGRAM` command. Also, histogram ID's are declared in `USER_his`. We will explain how to declare histograms in `USER_his` later. Anyway, you can see following statements in `gisspec_f.f`.

```
c make spectrum & image
      If ( (2 .eq. sensor) .or. (3 .eq. sensor) ) then
         offset = sensor * 100
         Call Hf1(offset+00, real(pha), 1.0)
         Call Hf1(offset+10, real(pi), 1.0)
         Call Hf1(offset+20, real(rt), 1.0)
         Call Hf1(offset+30, real(rti), 1.0)
         Call Hf2(offset+40, real(rawx), real(rawy), 1.0)
         Call Hf2(offset+50, real(detx), real(dety), 1.0)
         Call Hf2(offset+60, real(skyx), real(skyy), 1.0)
      Endif
```

In this example

```
         Call Hf1(offset+00, real(pha), 1.0)
```

filles `1.0` into a bin corresponding to a value of `pha` in 1 dimensional histograms with histogram ID of 200 and 300 for GIS–2 and GIS–3, respectively. In the same way

---

[6] This keyword is reserved by the ASCA_ANL system. The reserved EVS keywords are listed in Appendix C.

[7] The value for `ASCA_ANL_QUIT` is defined in asca_anl.inc. See a file `Includes.inc` in your current directory.

[8] If you analyze FRF with `GISREAD/GISTUNE`, obtained value is read from FRF.

```
            Call Hf2(offset+40, real(rawx), real(rawy), 1.0)
```

fills `1.0` into a bin corresponding to values of `RAWX` and `RAWY` in 2 dimensional histograms with histogram ID of 240 and 340 for GIS–2 and GIS–3, respectively. See Appendix D for an detailed explanation of arguments of Hf1 and Hf2.

The ASCA_ANL requires two kinds of settings as following:

```
c set GISSPEC:OK for ASCA_ANL_OK
      Call EVSset('GISSPEC:OK')
      status = ASCA_ANL_OK
c
      Return
      End
```

However, you don't worry about this because `mkanlmoudule` creates these statements automatically.

3.2.3 Definition of Histogram

In order to fill a value into a histogram the histogram is booked preceedingly. To do this, `USER_his` subroutine is used. In our `GISSPEC_his` histograms for GIS spectra and GIS images are booked.

```
c Define Histogram
      Call Hbook1(200, 'GIS2 PHA', 1024, -0.5, 1023.5, 0.0)
      Call Hbook1(300, 'GIS3 PHA', 1024, -0.5, 1023.5, 0.0)
      ...
c RAWX/Y is 0-255
      Call Hbook2(240, 'GIS2 RAWX vs RAWY',
     &    256, -0.5, 255.5, 256, -0.5, 255.5, 0.0)
      Call Hbook2(340, 'GIS3 RAWX vs RAWY',
     &     256, -0.5, 255.5, 256, -0.5, 255.5, 0.0)
```

The first two calls of `Hbook1` declare 1 dimensional histograms titled `GIS2 PHA` and `GIS3 PHA` with histogram ID of 200 and 300, respectively. Both are defined as histograms covering `-0.5` to 1023.5 with 1024 bins. Another two calls of `Hbook2` declare 2 dimensional histograms titled `GIS2 RAWX vs RAWY` and `GIS3 RAWX vs RAWY` with histogram ID of 240 and 340, respectively. Remember that these ID's are the same as those used in `GISSPEC_ana`. See Appendix D for an detailed explanation of arguments of Hbook1 and Hbook2.

The last statement is to set `status` (only one argument of `USER_ana`).

```
      status = ASCA_ANL_OK
      Return
      End
```

Again, you don't worry about this because `mkanlmoudule` creates these statements automatically.

3.2.4 User Interface

Some modules have several parameters to be specified by users. To ask parameters to users, `USER_com` is used. In our example program `gisspec_f` an good example can be seen in `circreg_com`.

```
c ask user for sensor ID
         Call Intrd('Sensor to set (-1=end of set)', sensor)
         .....
c ask user for center and radius of selection region
         Call Fltrd('DETX of Center (pixel)', cx(sensor))
         Call Fltrd('DETY of Center (pixel)', cy(sensor))
         Call Fltrd('Radius (pixel)', r(sensor))
```

The first call of `Intrd` ask a user for a four-byte integer value with `Sensor to set (-1=end of set)` as a prompt. The subroutine `Intrd` also prints out a value in the four-byte integer variable `sensor` before this call as a default value(See §2.2). After returning from this call, a specified value is set to the variable `sensor`. In the same way a four-byte floating point variable can be asked as in the above example. The subroutine `Fltrd` does everything boring such as printing out a message to a user and treating a default value. These subroutines `Intrd` and `Fltrd` are in the CLI package.

There are many cases to ask a user for various types of values including a string, a logical value, and so on. Sometimes you want to ask a user for a command which defines an action of your module. These are comfortably done by subroutines in the CLI/COM packages. See the memo on CLI/COM in files `cli.mem` and `com.mem`[9] as a reference. One of good examples of COM package is a command line interface of such a kind in the ASCA_ANL. You already know that the list of command is printed with a prompt like `ANL: Select 1 Option` or `ANL: Select 1 or More, Then OK`, that a command can be abbreviated, and so on. All these are done by a subroutine `Inquire`[10] in the COM package. A complex user interface is realized by a simple call of `Inquire` in your module.

Note that the unified user interface is one of the most important features of the ASCA_ANL and you are strongly recommended to use CLI/COM packages at user interface in your own module.

3.2.5 Initialization and Termination

In some applications we need to initialize a module before the analysis or to do something before the program is terminated. `USER_init` and `USER_exit` are reserved for such actions.

In `gisspec_f` only two initializations are done. First, EVS flags for the module are defined like this.

```
c EVS
      Call EVSdef('GISSPEC:BEGIN')
      Call EVSdef('GISSPEC:ENTRY')
      Call EVSdef('GISSPEC:OK')
```

Second, a version number of the module is put with `BNKput`.

```
      Call ANL_put_version('GISSPEC' , 'version 4.0')
```

These two are all you have to do in `USER_init` and are automatically generated by `mkanlmodule`. Of course, you can put other initialization in `USER_init`, for example, you can define other EVS flags or BNK keywords[11]. In most cases, however, you don't have to add anything to `USER_init`.

In `GISSPEC_exit` we can see nothing but setting a value to `status`.

```
      Subroutine GISSPEC_exit (status)
      Implicit NONE
c output
      Integer status
c include
      Include 'Includes.inc'
c
      status = ASCA_ANL_OK
      Return
      End
```

This means nothing is done for the module `GISSPEC` at the termination.

Again, in most cases you don't have to add anything to `USER_exit`.

---

[9] These are stored in a directory doc/ in the distribution kit. Ask a person who installed the ASCA_ANL in your computer where the directory is.

[10] See the memo `com.mem` for details.

[11] You have to be careful to define your own EVS/BNK keywords. See §4.3.1 for detailed explanations.

Note for initialization of local variale

Now you may plan to initialize variables for the module in `USER_init`. But, wait a minute. Does your module work O.K.?

In the ASCA_ANL `USER_init` is NOT necessarily called at first among seven subroutines required for a module, *i.e.*, `USER_com` and `USER_his` can be called before `USER_init` (See Figure 4.1). Therefore if you set a default value to a parameter in `USER_init` and if the parameter can be modified with a `MODIFY_PARAM` command, you will have an unexpected result in case that you modify the parameter. In this case the parameter is always set to the default value set in `USER_init` because `USER_init` is called after a call of `USER_com`.

To avoid this kind of problem, initialization of variables have to be done with another method. If you write a module in FORTRAN, you may use a block data initialization. If in C, you had better use a static global variable for such a parameter. In both cases the initialization is performed at their compilation, so you can get an expected action of your module.

3.2.6 Opening and Closing Data Set

`USER_bgnrun` and `USER_endrun` are subroutines to open and close the data set, respectively. Thus, Most modules does not have to fill these subroutines. In `gisspec_f` these two are almost blank. Although `EVSset` in `GISSPEC_bgnrun` have to be done here, `mkanlmodule` creates following subroutines automatically.

```
      Subroutine GISSPEC_bgnrun (status)
      Implicit NONE
c output:
      Integer status
c include
      Include 'Includes.inc'
c  EVS set
      Call EVSset('GISSPEC:BEGIN')
c
      status = ASCA_ANL_OK
      Return
      End

      Subroutine GISSPEC_endrun (status)
      Implicit NONE
c output
      Integer status
c include
      Include 'Includes.inc'
c
      status = ASCA_ANL_OK
      Return
      End
```

3.3 How to “Link” Your Module

To link your modules to the ASCA_ANL you have to write appropriate makefile for your program. However it is a little bit boring to create a makefile. So we recommend to use a makefile in the sample as a template. If you follow our recommendation, it is better for you to know what is happening while compiling and linking your modules. In this section we show how to link your modules from how to create makefile for the ASCA_ANL to what is happening in linking your modules.

3.3.1 Makefile in the Sample

Makefile for samples consist of three files, that is, `Makefile`, `Makefile.tmpl`, and `Includes.make`. `Makefile` contains usual rules to “make” your program, and `Makefile.tmpl` includes macro definitions, suffix rules, and so on. A directory structure of your computer system is described in `Includes.make` which is arranged by a person who installed the ASCA_ANL.

Since macros and suffix rules written in `Makefile.tmpl` are common and useful for most application of the ASCA_ANL, you can use this file as a tool box to "make" your module. You don't have to follow instructions in following sections when you "make" your module with `Makefile.tmpl`, because it automatically creates files required and compiles them. All you have to do is to create a file containing names of modules to be linked to your program. The rules to create this file are:

- the name of the file have to have an extension ".def"
- names of modules are separated by one carriage return
- the last character of this file is carriage return

To modify `Makefile` for your program a header of `Makefile.tmpl` (lines commented out at the beginning) is useful as a reference. You cut there, paste it to `Makefile`, modify macros to fit your program, and remove comment marks `#`. Here you are ready to "make" your program. Type,

```
% make
```

without your program name. Then `make` judges the OS, creates required files to link modules, compiles them, and links them. Finally you can get your own analysis program.

3.3.2 What is happening

When you use a makefile for samples as a template of a makefile for your program, you don't have to care about how to compile and link your program. However for those who want to know what is happening while samples are automatically compiled, we explain how to link your module to the ASCA_ANL and make an executable of your analysis program in following sections.

Following sections are written as instructions for you to link by yourself. Read and follow instructions if you are interested in. Note that in case of using sample makefile all these are automatically executed.

Overview

Before "making" your analysis program, you have to create one or two text files. The files includes several FORTRAN subroutines and C functions interfacing the ASCA_ANL system and your (or "ready-made") modules. Don't worry, there are commands in the ASCA_ANL package to create these files automatically. All you have to do is to specify the names of the modules.

The commands are `mkanlinit` and `mkanlCwrapper`. (Later in this section we will show you how to use them.) The "mkanlinit" creates a file containing C functions necessary for all the modules you want to link. Thus, you have to run the `mkanlinit` whenever you create a new analysis program.

The `mkanlCwrapper` creates a file containing a C functions[12] necessary for the modules written in C but not created by the command `mkanlmodule`[13]. Thus, if you don't have any C module, you don't have to run the `mkanlCwrapper`. But, if you have more than one module written in C, you have to run the "mkanlCwrapper". Note that you don't have to run `mkanlCwrapper` as long as you starts to write a module from a template created by `mkanlmodule`

To obtain an executable of your program, compile these interface files by a compiler in your system and link them to the ASCA_ANL together with the module you registered. An example of

---

[12] The functions just for interfacing C and FORTRAN.

[13] The `mkanlmodule` attaches the same interfacing functions at the end of a template. Create a template by `mkanlmodule` and see its end!

makefile is in the distribution kit of the ASCA_ANL and also in Appendix. You can easily create a makefile for your program by a slight change[14] of the sample makefile.

Now it is time to "make" it. You get your original analysis program. Enjoy!

mkanlinit

In order to construct your analysis program you have to register modules you use in the program. The mkanlinit creates a text file which contains a function `main()`' and a few C functions to register the modules. The text file usually has a name of `anlinit.c`, but you can specify it as you desired.

Anyway run mkanlinit first. The mkanlinit is in a distribution kit of ASCA_ANL. Please ask a system manager of your computer system where the mkanlinit is installed.

You will be asked for a name of the module you want to register.

```
Enter module name or *command [ 1]:  ?
```

If you want to register a module named `FITSREAD` at first, reply `FITSREAD` to the question.

```
Enter module name or *command [ 1]:  ? FITSREAD
Enter module name or *command [ 2]:  ?
```

If you also want to register a module named `USER`, reply `USER` to the next question.

```
Enter module name or *command [ 2]:  ? USER
Enter module name or *command [ 3]:  ?
```

In this way you can register as many modules as you need.

To finish specifying the module name press a return key without typing any name. Then you will see the module names to be registered and be asked for a name of an output file, *i.e.* the text file which contains a few FORTRAN subroutines to register the modules.

```
Enter module name or *command [ 3]:  ?

 Following modules are to be linked to ASCA_ANL:
     [ 1] FITSREAD
     [ 2] USER

Output file: anlinit.c ?
```

If you accept a default name `anlinit.c`, press return key. If you want to specify another name, reply the name to the question.

In case that a file with the same name already exists you will be confirmed if you intend to delete the old file or not.

```
'anlinit.c' already exists, delete (Y/N): N ?
```

If you reply `Y`, the old file will be deleted and the output file will be created. If you reply `N`, the old file will not be deleted and you will be asked for a file name again.

After replying all the questions above, the mkanlinit creates the text file containing the registration subroutines and terminates.

mkanlCwrapper

If you link modules written in C by your hand[15], you have to create another text file which contains several C functions[16] (called C wrappers) required for the C modules to be called by the

---

[14] For example, path name of the ASCA_ANL system in your computer system path name of libraries (*e.g.* CERN Library), name of the program you create, the modules you registered,...

[15] This means 'not by a command `mkanlmodule`' here.

[16] Written in "cfortran" format. The cfortran is a standard package in CERN Library interfacing C and FORTRAN.

ASCA_ANL written in FORTRAN. The text file usually has a name of `anlCwrapper.c`, but you can specify it as you desired.

Anyway run mkanlCwrapper first. The mkanlCwrapper is in a distribution kit of ASCA_ANL. Please ask a system manager of your computer system where the mkanlCwrapper is installed.

You will be asked for a name of the C module you want to link.

```
Enter module name [ 1]:  ?
```

If you want to link a module named `User`, reply `User` to the question.

```
Enter module name [ 1]:  ? User
Enter module name [ 2]:  ?
```

Notice that this specification is case-sensitive. You have to specify the exact name in C program. This fact also means you have to use the same case for all the entries. For example, once you make an entry `User_init`, you can not name a communication routine `USER_com` but you have to name it `User_com`.

To finish specifying the module name, press return key without typing any name. Then you will see the C module names to be linked to the ASCA_ANL and be asked for a name of an output file, *i.e.* the text file which contains several C wrappers for the ASCA_ANL.

```
Enter module name [ 2]:  ?

 Following modules (written in C) are to be linked to ASCA_ANL:
     [ 1] User

Output file: anlCwrapper.c ?
```

If you accept a default name `anlCwrapper.c`, press return key. If you want to specify another name, reply the name to the question.

In case that a file with the same name already exists you will be confirmed if you intend to delete the old file or not.

```
'anlCwrapper.c' already exists, delete (Y/N): N ?
```

If you reply `Y`, the old file will be deleted and the output file will be created. If you reply `N`, the old file will not be deleted and you will be asked for a file name again.

After replying all the questions above, the mkanlCwrapper creates the text file containing the C wrappers and terminates.

Chapter 4

Reference for Expert

4.1 Module and Command

A module is a unit of an analysis program with the ASCA_ANL. A user can create various analysis programs by combining modules. A module consists of seven subroutines. Each subroutine takes charge of a part of tasks of the module and a role of each subroutine is defined by the ASCA_ANL system. These subroutines are called by the ASCA_ANL system when corresponding commands are issued.

Table4.1 summarizes names of the subroutines, related commands, and roles which the subroutines should play. In Table4.2 and 4.3 we summarize commands of the ASCA_ANL program with a related subroutine in a module. In Table4.1, 4.2, and 4.3 we assume a module's name is ‘`USER` for clarity.

Name	Related Command	Role	Called When
`USER_init`	`ANALYZE_DATA`	Initialization	Jumping into the second stage of analysis, *i.e.*, after entering the command `ANALYZE_DATA` and before new command menu (includes `READ_DATA`, etc.) showing up.
`USER_com`	`MODIFY_PARAM`	User Interface	a module `USER` is selected in `MODIFY_PARAM` command.
`USER_his`	`BOOK_HISTOGRAM`	Histogram Booking	a module `USER` is selected in `BOOK_HISTOGRAM` command.
`USER_bgnrun`	`READ_DATA`	Data Set Opening	a data set is open.
`USER_ana`	`READ_DATA`	Event Analysis	an event are read from a data set.
`USER_endrun`	`READ_DATA`	Data Set Closing	a data set is closed.
`USER_exit`	`EXIT`	Termination	`EXIT` command is invoked.

Table 4.1: Subroutines composing a module

The relationship between the ASCA_ANL command and subroutines in a module are illustrated in Figure 4.1 In Figure4.1 each subroutine is indicated in a doubled oval. For example, `USER_init` should be in a doubled oval named “`_init`” entries, and so on. In addition, in each oval there are subroutines of one kind for some of the linked modules. In case of **`gisspec_f`**, subroutines `FITSREAD_init`, `CIRCREG_init`, and `GISSPEC_init` is in “`_init`” entries in this order[1].

---

[1] Exactly speaking, this order is defined by a command `DEFINE_ANALYSIS`. See §2.2 as reference.

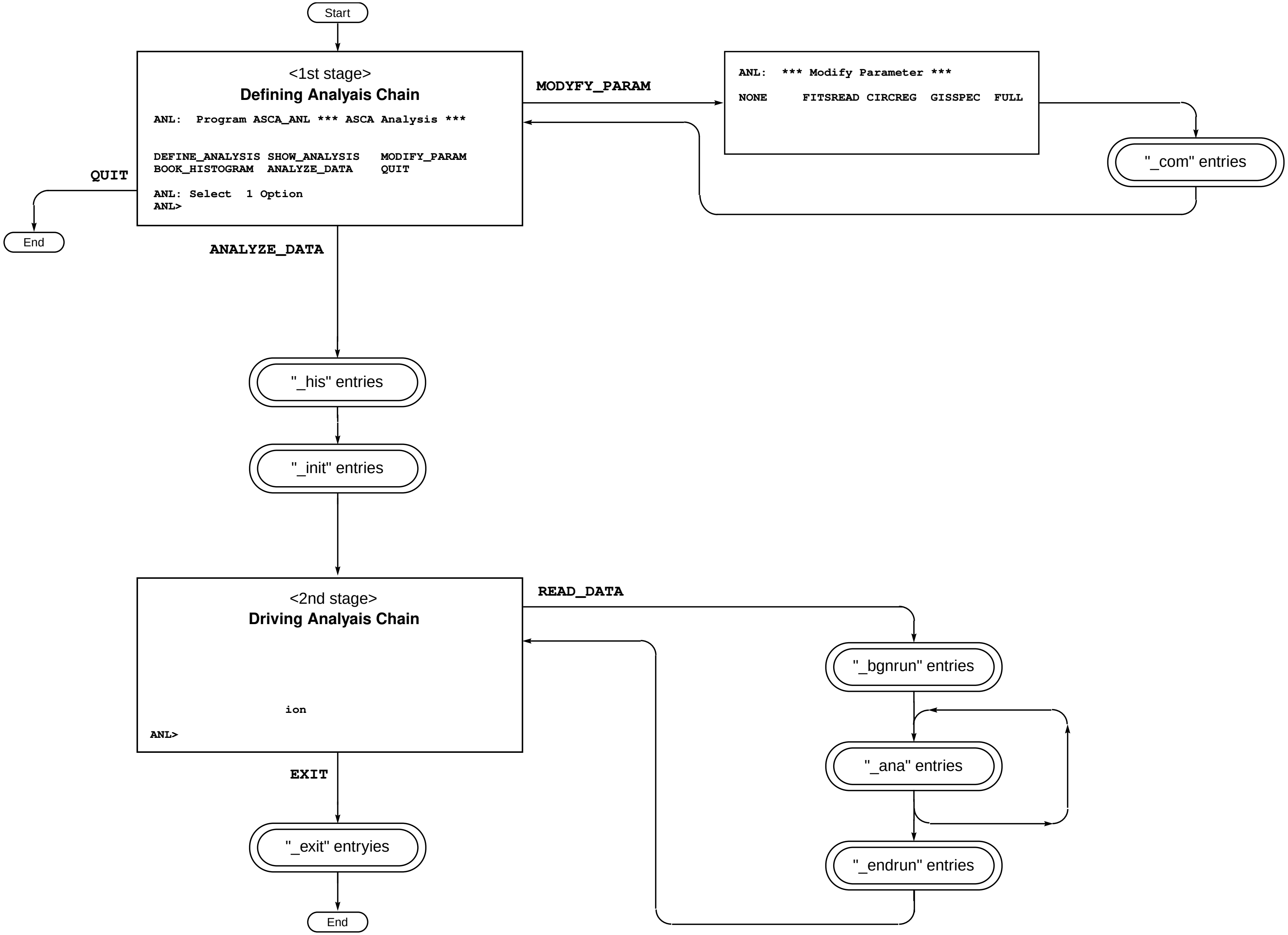


Figure 4.1: Schematic drawing of relationship between the ASCA_ANL command and subroutines in a module

Command	Related subroutine	Function
`DEFINE_ANALYSIS`	None	Define an analysis chain
`SHOW_ANALYSIS`	None	Show a currently defined analysis chain
`MODIFY_PARAM`	`USER_com`	Call user interface subroutines of modules
`BOOK_HISTOGRAM`	`USER_his`	Register modules for which histograms will be booked
`ANALYZE_DATA`	`USER_his`, `USER_init`	Book histograms for specified modules and jump to the second stage of the analysis
`QUIT`	None	Quit from the program without analyzing data

Table 4.2: Commands in the first stage (defining an analysis chain) on an analysis program

Command	Related subroutine	Function
`READ_DATA`	`USER_bgnrun`, `USER_ana`, and `USER_endrun`	Open data set and start analysis
`SAVE_HIST`	None	Save resultant histograms into an HBOOK file
`CLEAR_HIST`	None	Clear histograms
`SHOW_STATUS`	None	Print summary of EVS flags
`RESET_STATUS`	None	Clear summary of EVS flags
`EXIT`	`USER_exit`	Exit from the program

Table 4.3: Commands in the second stage (driving an analysis chain) on an analysis program

See Chapter7 or Chapter8 for details.

4.2 Analysis Chain

An analysis sequence with the ASCA_ANL is called an "analysis chain". The analysis chain consists of several modules, a root module to read data (See §4.4), some filter modules to filter events, and an (or a few) analysis module(s) to accumulate events to histograms. In this flow you may need to control an analysis flow to eliminate "bad" events.

To control an analysis flow, each module can receive/give information from/to the ASCA_ANL system through arguments of a `USER_ana` entry. In this section we summarize how to contact with the ASCA_ANL system with the arguments.

4.2.1 Flow Control

`USER_ana` entry has three arguments. The first argument is `nevent`, the second is `eventid`, and the third is `status`.

`nevent`

`nevent` is the number of events which a root module (See §4.4) reads successfully, or the number of iterations of an event loop. Therefore `nevent` starts from 1 and an analysis is terminated when `nevent` reaches the number of events specified at the beginning of event loop (See §2.2).

```
Number of events (-1=all, 0=exit): -1 ? 100
```

In this case `nevent` takes a value from 1 to 100 successively.

This is an input argument. Do not change this value in a module.

`eventid`

`eventid` shows a characteristic of a current event. There are two types of event at present, *i.e.*, an observation event and a pseudo event (See §4.2.2). If a current event is an observation event,

`eventid` = `EVENTID_OBS`,

and if a current event is a pseudo event,

`eventid` = `EVENTID_PSEUDO`.

This is an input argument. Do not change this value in a module.

`status`

`status` is a flag to control an analysis flow. The ASCA_ANL system evaluates `status` right after returning from a call of a module and determine the stream of an analysis flow so that you can control an analysis flow by setting a value to `status`.

A flow control can be specified from three points of view, whether a current event should be passed to downstream modules, whether the system should continue to read events from a current data set, and whether current status of EVS flags should be accumulated[2] (event counting). In Table 4.4 control values (values to be set to `status`) and their meanings are listed.

		Flow control		
Value	Additivity	Discard event	Discard data set	Disable to accumulate
ASCA_ANL_OK	No	×	×	×
ASCA_ANL_SKIP	No	○	×	×
ASCA_ANL_QUIT	No	○	○	○
ANL_DISCARD	Yes	○	×	×
ANL_ENDLOOP	Yes	×	○	×
ANL_NOCOUNT	Yes	×	×	○

Table 4.4: List of control values

There are two types of control values, additive values and non-additive (or classic) ones.

Additive values can be added to synthesize a desired flow control.

```
status = ANL_DISCARD + ANL_ENDLOOP
```

In this example a current event is discarded (*i.e.*, not be passed to downstream modules), a current data set is also discarded (*i.e.* remaining events in the data set is ignored), but current EVS flags are accumulated.

On the other hand, classic values can NOT be added. You only can set one value to `status`.

```
status = ASCA_ANL_QUIT
```

`status` is an output argument. Set a value before returning from a module and do not care about a value when a module is called.

4.2.2 Pseudo Event

An analysis program with the ASCA_ANL sometimes includes filter modules which filter events read from a data set. In such a case it can be difficult to measure an exposure time because the total exposure may be affected by all of the filters. If you have a "clock pulse" in the analysis

---

[2] EVS flags accumulated in an analysis is shown at the end of an analysis (printed just after `EXIT` command is invoked) or when `SHOW_STATUS` command is invoked.

system, however, you can easily measure the exposure time after fintering by counting the clock pulse after the filters.

A pseudo event is an artificial event which was introduced to calculate exposure time in a simple way. It is generated by a root module (See §4.4) at the top of an analysis chain and is filtered out by filtering modules (if so designed). To distinguish pseudo events from a normal events, several indicators work in a special way. Table4.5 summarizes the special indications for pseudo events.

Indicator	Attribution	Value for Pseudo Event	Value for Normal Event
event ID	Argument of `USER_ana`	`EVENTID_PSEUDO`	`EVENTID_OBS`
`ANL:SENSOR`	BNK keyword	`SENSOR_PSEUDO`	sensor ID
`SIS:PSEUDO Event`	EVS flag	"true" for SIS exposure	"false"
`GIS:PSEUDO Event`	EVS flag	"true" for GIS exposure	"false"
`SIS:SIS0 Pseudo`	EVS flag	"true" for SIS0 exposure	"false"
`SIS:SIS1 Pseudo`	EVS flag	"true" for SIS1 exposure	"false"
`GIS:GIS2 Pseudo`	EVS flag	"true" for GIS2 exposure	"false"
`GIS:GIS3 Pseudo`	EVS flag	"true" for GIS3 exposure	"false"

Table 4.5: Indicators of pseudo events

4.3 Data I/O

In order to get information on each event, BNK and EVS are available in a module. BNK provides a simple data bank system, which enables us to get and put the event data (*e.g.*, `ASCATIME`, `PI`, etc.). EVS manages flags related to each event and sums them up. Both BNK and EVS uses their keywords as pointers to the data and the flag. On the other hand, results of an analysis are stored in histograms by HBOOK subroutines (*e.g.*, `HF1` and `HF2`)[3]. . HBOOK provides us many useful subroutines which handles histograms.

4.3.1 BNK/EVS Keywords

The ASCA_ANL system defines a group of BNK keywords which are available for any module. Most of them are related to photon information, such as `ASCATIME` and `PI`. Also, some EVS keywords are reserved for common usage. Such keywords include information on a bit rate or a data mode. In Table4.6 and Table4.7 we summarize BNK and EVS keywords reserved by the system, respectively.

Note for user-defined BNK/EVS keyword

A user module can define a new BNK/EVS keyword in `USER_init`. The name of the keyword have to have an module's name like `USER:something` to avoid a possible overlapping of a keyword name. This restriction also affects how to refer such a data which other modules define and BNKput. Once you write a module `USER2` including BNKget like:

```
Call BNKget('USER:something', sizeof(something), size, something),
```

the module `USER2` ALWAYS have to be linked with the module `USER` and causes a fatal error in running when `USER` is not linked.

In order to go through this problem, a few choices are possible.

1. Change the module `USER2` not to refer `USER:something`.
   This is the simplest solution but is not always possible.

BNK keyword	aliases		storage		FITSREAD		SISREAD			SISTUNE			GISREAD		GISTUNE	
name	SIS	GIS	type	size	SIS	GIS	FNT	BRT	FST	FNT	BRT	FST	PH	MPC	PH	MPC
ANL:TIME	SIS:TIME	GIS:TIME	Real*8	8	O	O	O	O	O				O	O		
ANL:RAWX	SIS:RAWX	GIS:RAWX	Integer	4	O	O	O	O	O				O			
ANL:RAWY	SIS:RAWY	GIS:RAWY	Integer	4	O	O	O	O	O				O			
ANL:DETX	SIS:DETX	GIS:DETX	Integer	4	O	O				O	O	O			O	
ANL:DETY	SIS:DETY	GIS:DETY	Integer	4	O	O				O	O	O			O	
ANL:X	SIS:X	GIS:X	Integer	4	O	O				O	O	O			O	
ANL:Y	SIS:Y	GIS:Y	Integer	4	O	O				O	O	O			O	
ANL:PHA	SIS:PHA	GIS:PHA	Integer	4	O	O		O	O				O			
ANL:PI	SIS:PI	GIS:PI	Integer	4	O	O									O	
ANL:SENSOR			Integer	4	O	O	O	O	O				O	O		
ANL:FILENAME			Character	*	O	O	O	O	O				O	O		
ANL:PSEUDO_INTERVAL			Real*8	8	O	O	O	O	O				O	O		
ANL:EULER			Real*8	8x3	O	O	O	O	O				O	O		
ANL:SKYREF			Real*8	8x3	O	O	O	O	O				O	O		
SIS:PHAS			Integer	4x9	O		O									
SIS:CCDID			Integer	4	O		O	O	O							
SIS:GRADE			Integer	4	O			O	O							
GIS:PHAS			Integer	**		O								O		
GIS:RISE_TIME			Integer	4		O							O			
GIS:RTI			Integer	4		O									O	
GIS:SPREAD			Integer	4		O							O			

* maximum size of a file name in ANL:FILENAME is 256

** size of GIS:PHAS ranges 4x (1-256)

Table 4.6: BNK keywords

EVS keyword	FITSREAD	SISREAD	GISREAD
`ANL:BIT HIGH`	○	○	○
`ANL:BIT MEDIUM`	○	○	○
`ANL:BIT LOW`	○	○	○
`SIS:FAINT mode`	○	○	
`SIS:BRIGHT mode`	○	○	
`SIS:FAST mode`	○	○	
`SIS:BRIGHT2 mode`	○		
`GIS:PH mode`	○		○
`GIS:MPC mode`	○		○
`SIS:SIS0 Event`	○	○	
`SIS:SIS1 Event`	○	○	
`GIS:GIS2 Event`	○		○
`GIS:GIS3 Event`	○		○
`SIS:PSEUDO Event`	○	○	
`GIS:PSEUDO Event`	○		○
`SIS:SIS0 Pseudo`	○	○	
`SIS:SIS1 Pseudo`	○	○	
`SIS:GIS2 Pseudo`	○		○
`SIS:GIS3 Pseudo`	○		○

Table 4.7: EVS flags

2. Check existence of the module `USER` before BNKget in the module `USER2`.
   To check the existence, we can use `ANL_get_chain_info`, or one of utility routines supplied with the ASCA_ANL system. See Chapter 5 for details.

In general it is not recommended to design a module which is largely depending on an other module's existence. Such a tight relationship between modules may reduce convenience of "modularization" and value of your module as a software resource. In some case it indicates a possibility that you can merge their functions into one module. In any case please check necessity and validity of dividing a function into two modules before you design such a module.

4.3.2 BNKF/EVSF — Faster BNK/EVS —

When data in BNK or EVS is accessed (*e.g.*, when `BNKget` or `EVSset` is called), BNK and EVS searches a specified keyword in an internal keywords table, find a place in which specified data is stored, and return the stored value. Because of these searching keywords data I/O with BNK or EVS can be a problem by reducing a speed on its execution.

To avoid this, following four subroutines are implemented.

```
Subroutine BNKFput(key, index, n, iarray)
Character*(*) key          ! input
Integer index              ! input/output
Integer n                  ! input
Integer iarray             ! output

Subroutine BNKFget(key, index, size, used, iarray)
Character*(*) key          ! input
Integer index              ! input/output
Integer size               ! input
Integer used               ! output
```

```
Integer iarray            ! output

Logical Function EVSF(key,index)
Character*(*) key         ! input
Integer index             ! input/output

Subroutine EVSset(key,index)
Character*(*) key         ! input
Integer index             ! input/output
```

These subroutines search a place of specified data when it is called with `index = 0` and return the place as `index`. When you call these with the returned `index`, these return a value in the place specified by `index` without searching the keyword to find the stored data. In this way we can reduce an overhead of the call of these subroutines.

Typical usage of these subroutines will be followings. In FORTRAN,

```
      Integer sensor, size
      Integer index1/0/, index2/0/, index3/0/, index4/0/
      Save index1, index2, index3, index4

      Call BNKFput('ANL:SENSOR', index1, sizeof(sensor), sensor)
      Call BNKFget('ANL:SENSOR', index2, sizeof(sensor),
     &             size, sensor)
      Call EVSFset('USER:ENTRY', index3)
      If ( EVSF('ANL:GIS2 Pseudo', index4) )
     &   Write(*,*) 'GIS2 Pseudo'
```

and in C,

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cfortran.h>
#include <bnk.h>
#include <evs.h>

        int sensor, size;
        static int index1=0, index2=0, index3=0, index4=0;

        BNKFPUT("ANL:SENSOR", index1, sizeof(sensor), &sensor);
        BNKFGET("ANL:SENSOR", index2, sizeof(sensor), size, &sensor);
        EVSFSET("USER:ENTRY", index3);
        if ( EVSF("ANL:GIS2 Pseudo", index4) ) printf("GIS2 Pseudo");
```

4.3.3 Support Macros for BNKF/EVSF

For C language people there are another way of making use of faster BNK/EVS system. The macro definition file `bnkf.h` and `evsf.h` are in the ASCA_ANL distribution kit.

include/bnkf.h:

```
#define BNKFGETM(key,size,used,value)   {\
        static int index = 0;\
        int vsize = size;\
        bnkfget_(key, &index, &vsize, &used, value, sizeof(key)-1);\
}

#define BNKFPUTM(key,size,value)        {\
        static int index = 0;\
        int vsize = size;\
        bnkfput_(key, &index, &vsize, value, sizeof(key)-1);\
}
```

include/evsf.h:

```
#define EVSFM(key,index)        evsf_(key, &(index), sizeof(key)-1)
#define EVSFSETM(key)   {\
        static int index = 0;\
        evsfset_(key, &index, sizeof(key)-1);\
}
```

Typical usage will be a following.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cfortran.h>
#include <bnk.h>
#include <bnkf.h>       /* <--- additional! */
#include <evs.h>
#include <evsf.h>       /* <--- additional! */

        int sensor, size;
        static int index = 0;

        BNKFPUTM("ANL:SENSOR", sizeof(sensor), &sensor);
        BNKFGETM("ANL:SENSOR", sizeof(sensor), size, &sensor);
        EVSFSETM("USER:ENTRY");
        if ( EVSFM("ANL:GIS2 Pseudo", index) ) printf("GIS2 Pseudo");
```

CAUTION

Because support macros in `bnkf.h` and `evsf.h` are NOT functions, you have to pay a special attention in writing a module. You have to write macros as many times as the number of data you want to access.

If you write a following code to get four data, "ANL:SENSOR", "ANL:DETX", "ANL:DETY", and "ANL:PI", it does NOT work.

```
int i, value[4];
static char keyword[4][20] = {"ANL:SENSOR","ANL:DETX","ANL:DETY","ANL:PI"};
for (i=0;i<4;i++) {
        BNKFGETM(keyword[i], sizeof(value[i]), size, &value[i]);
}
```

In above case a macro `BNKFGETM` is expanded as one line so that only one internal index is reserved. If you need to get four data with `BNKFGETM`, you have to write `BNKGETM` four times like this.

```
int value[4];
BNKFGETM("ANL:SENSOR", sizeof(value[0]), size, &value[0]);
BNKFGETM("ANL:DETX", sizeof(value[1]), size, &value[1]);
BNKFGETM("ANL:DETY", sizeof(value[2]), size, &value[2]);
BNKFGETM("ANL:PI", sizeof(value[3]), size, &value[3]);
```

4.3.4 Buffer Size of BNK and HBOOK

Both BNK and HBOOK allocate a certain size of memory to store the data. Hereafter we call the memory area "BNK buffer" and "HBOOK buffer", respectively. Because the size of these buffers are defined so large that most analysis can go well. In some applications, however, you may need lager size of memory to be allocated for BNK data or histograms.

The size of buffer can be specified in `mkanlinit`. When you reply a following question with "*", `mkanlinit` recognizes it as a beginning of a parameter name to be set.

```
Enter module name or *command [ 1]:  ?
```

Therefore type as followings if you want to set the size of HBOOK buffer to 2000000.

```
Enter module name or *command [ 1]:  ? *HBOOK_BUF_SIZE 2000000
Enter module name or *command [ 1]:  ?
```

Note that the unit of the size is four bytes so that “2000000” means 8000000 bytes.

Again abbreviations are valid here[3]. Also, any space and carriage return after the “*” are ignored. Thus, Following replies make no difference.

```
Enter module name or *command [ 1]:  ? *HBOOK 2000000
Enter module name or *command [ 1]:  ?

Enter module name or *command [ 1]:  ? *HBOOK_BU 2000000
Enter module name or *command [ 1]:  ?

Enter module name or *command [ 1]:  ? * HBOOK_BUF_SIZE 2000000
Enter module name or *command [ 1]:  ?

Enter module name or *command [ 1]:  ? *
Command: HBOOK_BUF_SIZE ? HBOOK_BUF_SIZE
HBOOK_BUF_SIZE: 2200000 ? 2000000
Enter module name or *command [ 1]:  ?
```

In Table4.8 we summarize parameters which you can set in `mkanlinit`.

Parameter	Unit	Remark
`HBOOK_BUF_SIZE`	4 bytes	size of HBOOK buffer
`BNK_BUF_SIZE`	4 bytes	size of BNK buffer

Table 4.8: Parameters to be set in `mkanlinit`

Although the sizes of both BNK and HBOOK buffer can be expanded with this feature of `mkanlinit`, you must be careful to determin the size to be set. Even if you allocate a large size of memory for histograms (HBOOK buffer), it is no use if *Display45* can not read in your resultant histograms. So there seems to make no sense to expand the size more than that allocated in *Display45*[4]. For the BNK buffer, getting large size of memory of the data area may cause a large overhead of copying the data when the data is accessed (with `BNKget`, etc.). Perhaps you can find another solution to realize your design without expanding the BNK buffer to an unrealistic size.

4.4 Root Module

A root module[5] is specially designed to read events from a data set and send them to downstream modules. Therefore it is required that a root module have to exist in an analysis chain, that only one root module are allowed in a chain, and that it should be at the top of a chain.

Because of its special role in a chain there are several additional rules in writing it. Let us assume you are writing a root module named `ROOT`. Followings are required rules, *i.e.*, ANY root module have to satisfy following rules.

1. Call `ANL_initroot` in `ROOT_init`.
   `ANL_initroot` defines common BNK/EVS keywords. To call this in FORTRAN, just say,

```
...
      Subroutine ROOT_init (status)
      Implicit None
...
      Integer st
...
c Common BNK/EVS
      Call ANL_initroot(st)
...
```

---

[3] But we do NOT recommend abbreviate too much.

[4] Ask an installer of *Display45* for the size.

[5] `FITSREAD, SISREAD, and GISREAD are supplied with the ASCA_ANL.`

and in C,

```
...
#include <anl_initroot.h>
...
void
root_init(int *status)
{
   int st;
   ...
/* Common BNK/EVS */
   ANL_INITROOT(st);
   ...
 }
```

the return status `st` of `ANL_initroot` is reserved for the future and means nothing at present.

2. Set "event ID".
   An event ID is the second argument of `ROOT_ana` entry. The values to be set are:

   event ID = `EVENTID_PSEUDO` in case of a pseudo event
   event ID = `EVENTID_OBS` in case of an observation data

   If you write a root module in C, special procedures are required to set an event ID.

   (a) Change an definition of function `ROOT_ana`.
   The second argument must change to a pointer to an integer.

   ```
   void
   root_ana(int nevent, int *eventid, int *status)
   ```

   (b) Change a C wrapper for `ROOT_ana` entry.
   The fifth argument[6] of the C wrapper must change from `INT` to `PINT`.

   ```
   FCALLSCSUB3(root_ana,ROOT_ANA,root_ana,INT,PINT,PINT)
   ```

   (c) Change an declaration of function `ROOT_ana` (if any).

   ```
   void root_ana(int, int *, int *);
   ```

In addition, there are recommended feature of a root module.

1. BNKput Euler angles.
   For attitude correction we need to know an attitude as Euler angles. To unify how to access an attitude information, *i.e.*, how to get Euler angles at the moment, a root module are recommended to BNKput Euler angles for downstream modules. BNK keyword's name is "`ANL:EULER`" (See §4.3.1).

2. Generate pseudo events.
   Pseudo events are very useful to measure an exposure time of an observation. Many users make use of pseudo events to do that, a root module is strongly recommended to generate pseudo events. Also, a root module have to BNKput an interval of pseudo events in second with a BNK keyword "`ANL:PSEUDO_INTERVAL`" (See §4.3.1).

3. BNKput a current file name.
   In most applications the data set is provided as a event file. A root module is recommended to BNKput its name for downstream modules. BNK keyword is "`ANL:FILENAME`" (See §4.3.1).

---

[6] Or the second from the right

For downstream modules, of course, a root module have to as many information of events in a data set as possible. For example, a root module have to BNKput various information on a current event, *e.g.*, `ASCATIME` to "`ANL:TIME`", `DETX` to "`ANL:DETX`", and `PI` to "`ANL:PI`". Also, it must to EVSset various flags, *e.g.*, "`SIS:SIS0 Event`" for an observation event of SIS0, "`SIS:FAINT`" for an SIS faint mode event, and "`ANL:BIT HIGH`" in case of an event taken during high bit rate (See §4.3.1). That means design of a root module is very important for the ASCA_ANL system. Please note that a root module may determine quality of an analysis in some cases. In the end we recommend to consult a member of ASCA_ANL Working Group before designing a root module.

Chapter 5

Utility Routines

Here is a brief description of the utility routines supplied with the ASCA_ANL. With the utility routines, you can access many kinds of information required in programming a new module. Some functions can be realized without using the utility routines, however, you are recommended to use the utility routines.

In this section we briefly introduce functions of the utility routines. All the utility routines here are supplied in a library `libANL_UTIL.a`. The library are automatically created by installing the ASCA_ANL (see §A).

The following functions are supplied.

Name	Function	Ref.
`ANL_readHK_register`	register HK items	§5.1
`ANL_readHK`	get an HK value from HK files or FRF	§5.1
`ANL_readGISmoni`	count up GIS/RBM monitor counts in a specified time interval	§5.2
`ANL_orbit`	get orbital and attitude information	§5.3
`ANL_GETLUN`	get an unoccupied logical unit number	§5.4
`ANL_FREELUN`	set free the occupied logical unit number	§5.4
`ANL_GET_CHAIN_INFO`	get module names in an analysis chain	§5.5
`ANL_CURRENT_CALL`	get the order of a module called currently in an analysis chain	§5.5

5.1 ANL_readHK

`ANL_readHK` is a function to get HK values from HK files or FRF.

First, if you get HK values from HK files, set a name of a directory in which HK files are stored to an environment variable `FHKDIR`. If you have HK files in several directories, you can set names of the directories separated by a delimiter ':'. If you set nothing to `FHKDIR`, HK files are searched from the current directory. If you get HK values from FRF, you need not set `FHKDIR` but must include `GISREAD` or `SISREAD` in an analysis chain.

Second, call `ANL_readHK_register` to register HK items.

```
int ANL_readHK_register(char *HKitem);
```

`HKitem` is a name of HK item you want. `ANL_readHK` returns an ID for the HK item. If a error occurs, `ANL_readHK_register` returns a negative value.

Third, call `ANL_readHK` with `ascatime` and `HKid` to get a HK value you need.

```
int ANL_readHK(int HKid, double ascatime, double mtime[2]);
```

`HKid` is an integer value which `ANL_readHK_register` returned. `ANL_readHK` returns a HK value at `ascatime` which corresponds to `HKid`. Also, `ANL_readHK` returns the reference times in an array `mtime`. `mtime[0]` is a time stamp of the latest HK value before specified `ascatime` in a HK file and `mtime[1]` is a time stamp of the first HK value appeared after `ascatime` in a HK file. If a error occurs, `ANL_readHK` returns a negative value. Notice that it may not work well if you call `ANL_readHK_register` after calling `ANL_readHK`.

5.2 ANL_readGISmoni

`ANL_readGISmoni` is a function to count up GIS and/or RBM monitor counts in a specified time interval. Following shows the sample call of `ANL_readGISmoni`. You have to set integration times for **ALL** monitor counts individually. If you set zero to an integration time of a certain monitor count, `ANL_readGISmoni` does not count up the monitor count. In the following sample, RBM, H0 and H2 of GIS2/3 will be counted up each four seconds.

```
#incldue "anl_gismoni.h"

    double ascatime;    /* ASCATIME of interest */
    struct ANL_GISmoni gm;

    gm.rb.dt = 4;       /* RBM count */

    gm.g2.ld.dt = 0;    /* GIS2 LDHIT */
    gm.g2.l0.dt = 0;    /* GIS2 L0 */
    gm.g2.l1.dt = 0;    /* GIS2 L1 */
    gm.g2.l2.dt = 0;    /* GIS2 L2 */
    gm.g2.h0.dt = 4;    /* GIS2 H0 */
    gm.g2.h1.dt = 0;    /* GIS2 H1 */
    gm.g2.h2.dt = 4;    /* GIS2 H2 */
    gm.g2.ci.dt = 0;    /* GIS2 CPUIN */
    gm.g2.co.dt = 0;    /* GIS2 CPUOUT */
    gm.g2.to.dt = 0;    /* GIS2 TLMOUT */

    gm.g3.ld.dt = 0;    /* GIS3 LDHIT */
    gm.g3.l0.dt = 0;    /* GIS3 L0 */
    gm.g3.l1.dt = 0;    /* GIS3 L1 */
    gm.g3.l2.dt = 0;    /* GIS3 L2 */
    gm.g3.h0.dt = 4;    /* GIS3 H0 */
    gm.g3.h1.dt = 0;    /* GIS3 H1 */
    gm.g3.h2.dt = 4;    /* GIS3 H2 */
    gm.g3.ci.dt = 0;    /* GIS3 CPUIN */
    gm.g3.co.dt = 0;    /* GIS3 CPUOUT */
    gm.g3.to.dt = 0;    /* GIS3 TLMOUT */

    ANL_readGISmoni(ascatime, &gm);
```

Return values for RBM counts are, for example, stored in following variables:

`double gm.rb.t0`	Start time of integration
`double gm.rb.t1`	End time of integration
`double gm.rb.dt`	(`gm.rb.t1` − `gm.rb.t0`)
`int gm.rb.count`	RBM counts in the integration time

An instantaneous counting rate (c/s) of RBM counts at `ascatime` can be calculated by `(gm.rb.count/gm.rb.dt)`. Note that `gm.rb.dt` is set to 0.0 when `ascatime` is out of bounds.

At present, `ANL_readGISmoni` works only in a limited condition. Also, you have to pay careful attention to its usage. As a summary, you should note that following points in using `ANL_readGISmoni`:

- `ANL_readGISmoni` must be used with GISREAD or SISREAD.
  This function does not work with FITSREAD.
- `ANL_readGISmoni` can be called only from a C module.
- `ANL_readGISmoni` does not check that GIS or RBM are in the proper state (GIS/RBM/HV/CPU ON/OFF, HV level, etc).
- GIS monitor counts are not set in PCAL mode.
- TLMOUT is not set in MPC mode.
- It's no use to set integration time longer than 16 SF (64 s in bit-H),
  because GISREAD or SISREAD hold only 16 SF data in the memory.
- Most of monitor counts are output once in 4 frames which correspond to 4 s in bit-L.
  (RBM and TLMOUT 1 frame, and LDHIT 2 frames)
- Integration start and end are (`ascatime`−`dt`/2) and (`ascatime`+`dt`/2) in principle,
  but it can be shifted if corresponding SF are not in the memory.

5.3 ANL_orbit

`ANL_orbit` is a function to get orbital and attitude information, such as rigidity, elevation, and so on. A user can get the information by calling a function `ANL_orbit` with the ascatime. Note that `ANL_orbit` can be called only from a C module at present.

```
#include <anl_orbit.h>
   ...
   ANL_orbit(ascatime, &asca);
```

The function `ANL_orbit` calls atFunctions through functions in ascatool and the information is supplied by a structure variable, `*asca`. The types of variables are shown in Table 5.1 and, Table 5.2. If no error occurred, `ANL_orbit` returns `ANL_ORBIT_OK`. Otherwise, `ANL_orbit` returns `ANL_ORBIT_ERROR`.

Table 5.1: ANL_orbit parameter

parameter	unit	I/O	type
ascatime	sec	Input	double
*asca	—	Output	ANLorbit

5.4 ANL_GETLUN/ANL_FREELUN

If your module needs a logical unit number for a file I/O, use subroutines `ANL_GETLUN` and `ANL_FREELUN`. The subroutine `ANL_GETLUN` gives us an unoccupied logical unit number, and the subroutine `ANL_FREELUN` releases the logical number.

DO NOT use your favorite number for a logical unit number. That may cause fatal errors in file I/O. This means you may destroy data in science FITS file in such a case.

Followings are examples of calling these subroutines. In the example you get or release a logical unit number through an integer variable `lun`.

Table 5.2: structure member of ANLorbit

variable name	unit	type	remark
phi	degree	double	Z-Y-Z first Euler angle
theta	degree	double	Z-Y-Z second Euler angle
psi	degree	double	Z-Y-Z third Euler angle
RA	degree	double	Right Ascension of Target (J2000)
DEC	degree	double	Declination of Target (J2000)
height	km	double	altitude of the satellite from the earth surface
longitude	degree	double	longitude of the satellite on the earth surface, $0 \leq$ longitude < 360
latitude	degree	double	latitude of the satellite on the earth surface, $-90 \leq$ latitude ≤ 90
rigidity	GeV/c	float	cutoff rigidity
elevation	degree	double	elevation angle from the earth edge
earth	—	int	0:Sky 1:NightEarth 2:BrightEarth
day_night	—	int	1:Night else Day on the satellite
saa	—	int	1:in 0:out SAA

In FORTRAN

To get a logical unit number,

```
Integer*4 lun
...
Call ANL_getlun(lun)
```

To release the logical unit number,

```
Integer*4 lun
...
Call ANL_freelun(lun)
```

In C

To get a logical unit number,

```
#include <anl_misc.h>
   ...
   int lun;
   ...
   ANL_GETLUN(lun);
```

To release the logical unit number,

```
#include <anl_misc.h>
   ...
   int lun;
   ...
   ANL_FREELUN(lun);
```

Notice that an argument `lun` is NOT a pointer to an integer but an integer variable itself. This is because we use "cfortran" to interface C and FORTRAN. See a manual of "cfortran"[9] for details.

5.5 ANL_GET_CHAIN_INFO/ANL_CURRENT_CALL

You can get information on an analysis chain by calling a subroutine `ANL_GET_CHAIN_INFO` or a function `ANL_CURRENT_CALL`. The subroutine `ANL_GET_CHAIN_INFO` returns an array of character strings which contains module names in an analysis chain currently defined. The function `ANL_CURRENT_CALL` returns an integer which indicates the module currently called by the ASCA_ANL system. The number corresponds to the order of calling modules in a defined analysis chain[1].

Followings are examples of calling these subroutines. In the example you pass `MAXSIZE`, or a size of an array of character strings `chain`. The array `chain` is used to get module names. You also get `defined`, or the number of modules currently defined in the analysis chain, and `registered`, or the number of modules currently registered to the ASCA_ANL system.

In FORTRAN

To get information on the analysis chain,

```
Integer MAXSIZE
Parameter (MAXSIZE=10)
character*16 chain(MAXSIZE)
Integer defined, registered
...
Call ANL_get_chain_info(MAXSIZE, chain, defined, registered)
```

To know which module is currently called,

```
Integer current
Integer ANL_current_call
...
current = ANL_current_call
```

In C

To get information on the analysis chain,

```
#define MAXSIZE 10
    ...
    char chain[MAXSIZE][16];
    int defined, registered;
    ...
    ANL_GET_CHAIN_INFO(MAXSIZE, chain, defined, registered);
```

Notice that arguments `defined` and `registered` are NOT pointers to an integer but integer variables themselves. This is because we use "cfortran" to interface C and FORTRAN. See a manual of "cfortran"[9] for details.

To know which module is currently called,

```
int current;
...
current = ANL_CURRENT_CALL();
```

---

[1] For example, if you defined an analysis chain which includes three modules "A", "B", "C" in this order, `ANL_CURRENT_CALL` returns 2 while the system is calling the module "B".

Chapter 6

Built-in Modules

The ASCA_ANL has a few built-in modules to do common tasks[1] in a data analysis. The modules are designed[2] to avoid "a flood of software tools"[3] by unifying a tool for each task. Therefore, you are strongly recommended to use the built-in modules when you needs the same function as that of one of built-in module.

All the built-in modules here are listed in Table 6.1 and supplied as a library `libANL_MODULE.a`. The libraries are automatically created by installing the ASCA_ANL (see §A).

In this section we briefly introduce functions of the built-in modules.

Name	Function	Reference
`FITSREAD`	Read FITS file and extract the data for each photon	§6.1
`SISREAD`	Read FRF and extract the SIS raw data for each photon	§6.2
`SISTUNE`	Apply various calibration constants to the raw SIS data	§6.2
`GISREAD`	Read FRF and extract the GIS raw data for each photon	§6.3
`GISTUNE`	Apply various calibration constants to the raw GIS data	§6.3

Table 6.1: Functions of built-in modules in a library `libANL_MODULE.a`

IMPORTANT NOTE FOR VERSION 2.00 OR LATER

`SISREAD`, `SISTUNE`, `GISREAD`, and `GISTUNE` are NOT included in the ASCA_ANL package version 2.00 or later. Instead, these are supplied through ASCA_ANL_contrib[4].

6.1 FITSREAD

`FITSREAD` is one of the most up-stream modules. It reads FITS file and extracts the data for each photon by utilizing FITSIO routine. For the modules in the lower stream of the analysis sequence, `FITSREAD` puts data in the BNK data buffer. User module can access data by calling `BNKget` in their analysis routine. The BNK keywords are same as ones defined in the FITS file, except for the header "`GIS:`" for GIS and "`SIS:`" for SIS. BNK keywords and the size used in `FITSREAD` are summarized in Table 4.6.

---

[1] For example, a task of reading data from FITS file is commonly required in a data analysis.

[2] Clearly, as a result, the modules reduce your labor to create a new analysis program.

[3] This means different people make different tools with different algorithm, which leads to the different result for the same analysis.

[4] Read page ii for details on ASCA_ANL_contrib.

Pseudo Event

`FITSREAD` can generate pseudo events separated by a constant time interval. The time interval can be specified arbitrarily. You can distinguish between a observation event and a pseudo event using **`eventid`**, `EVS`, or a BNK keyword "`ANL:SENSOR`". You can calculate an exposure time with pseudo events.

Parameter Settings

You may specify how `FITSREAD` acts by setting parameters with `MODIFY_PARAM` command. In Table 6.2 we summarize commands in `MODIFY_PARAM` of `FITSREAD`.

Command	Function	Default
`SHOW`	Display current settings	—
`PSEUDO_INTERVAL`	Set time interval of pseudo events[a]	0.0
`EULER`	Set whether to BNKput Euler angles	No
`EXIT`	Exit from `MODIFY_PARAM`	—

---

[a] No pseudo event is generated if set to 0.0.

Table 6.2: Commands in `MODIFY_PARAM` of `FITSREAD`

6.2 SISREAD with SISTUNE

`SISREAD` is another most up-stream module that delivers data to other modules. It is based on the utility functions implemented in **`tlmeventsFRF`** written by K. Mitsuda (ISAS) and allows a user to access FRF file directly. Note that `SISREAD` only reads raw data such as `CCDID`, `PHA`, `RAWX`, etc. In order to calculate calibrated data, such as `DETX`, `X`, `SISTUNE` should be assigned as a down stream module. `SISTUNE` reads raw data and converts them into calibrated data based on calibration constants. `SISTUNE` also utilizes functions in **`tlmeventsFRF`**. Calibrated data are stored in BNK data buffer and passed to the down stream modules. The same keywords as `FITSREAD` are used for BNK in `SISREAD/SISTUNE`.

Pseudo Event

`SISREAD` can generate pseudo events at the beginning of each CCD frame. The interval can't be specified arbitrarily. Using these pseudo events, you can not only recognize CCD frames easily but also calculate an exposure time.

Parameter Settings

You may specify how `SISREAD` and `SISTUNE` acts by setting parameters with `MODIFY_PARAM` command. In Table 6.3 and 6.4 we summarize commands in `MODIFY_PARAM` of `SISREAD` and `SISTUNE`, respectively.

Command	Function	Default
`BitRate`	Select bit rate of the data	ANY
`Pseudo-Event`	Set whether to generate pseudo events[a]	OFF
`Euler-Angle`	Set whether to BNKput Euler angles	Yes
`Return-to-Menu`	Return to ASCA_ANL menu (Exit from `MODIFY_PARAM`)	—

---

[a] You can not specify its time interval

Table 6.3: Commands in `MODIFY_PARAM` of `SISREAD`

Command	Function	Default
`Teldef-File`	Sets a name of a teldef file	"-"
`Reference-Position`	Sets a reference point to convert coordinates	(0.0, 0.0)
`Return-to-Menu`	Return to ASCA_ANL menu (Exit from `MODIFY_PARAM`)	—

Table 6.4: Commands in `MODIFY_PARAM` of `SISTUNE`

6.3 GISREAD with GISTUNE

`GISREAD` is yet another most up-stream module that delivers data to other modules. It is based on the utility functions implemented in `GIPHS` written by Y. Ishisaki and allows a user to access FRF file directly. Note that `GISREAD` only reads raw data such as `PHA`, `RAWX`, `RISE_TIME`, etc. In order to correct these raw data `GISTUNE` should be assigned as a down stream module. `GISTUNE` obtains data and applies various calibration constants to the raw data. `GISTUNE` also utilizes functions in `GIPHS`. Corrected data is stored in BNK data buffer and passed to the down stream modules. The same keywords as `FITSREAD` are used for BNK in `GISREAD/GISTUNE`.

Pseudo Event

`GISREAD` can generate pseudo events separated by a constant time interval. The interval can be specified arbitrarily. You can distinguish between an observation event and a pseudo event using `eventid`, `EVS`, or a BNK keyword "`ANL:SENSOR`". You can calculate an exposure time with pseudo events.

Parameter Settings

You may specify how `GISREAD` and `GISTUNE` acts by setting parameters with `MODIFY_PARAM` command. In Table 6.5 and 6.7 we summarize commands in `MODIFY_PARAM` of `GISREAD` and `GISTUNE`, respectively.

Special BNK Keywords

`GISREAD` and `GISTUNE` define special BNK keywords showed in Table 6.6 and 6.8, respectively.

Special EVS Keywords

`GISTUNE` defines special EVS keywords showed in Table 6.9.

Command	Function	Default
`SHOW`	Display current settings	—
`GIS2`	Read GIS2 events	ON
`GIS3`	Read GIS3 events	ON
`GISMON`	Set whether GIS monitor counts should be counted by EVS	OFF
`EULER`	Set whether Euler angle should be BNKput	ON
`BITRATE`	Select bit rate of the data	HML
`POINTING`	Select data in pointing mode[a]	ON
`OBS`	Select data in observation mode	ON
`GISOK`	Select data only when GIS status is O.K.[b]	ON
`GISNORM`	Select data only when GIS mode is normal[c]	ON
`GISSPD`	Read data with GIS spread discr. flag ON/OFF/BOTH	BOTH
`GISCNT`	Set min./max. value of GIS count rate (c/s/super-frame)	0.0 – 0.0
`RBMCNT`	Set min./max. value of RBM count rate (c/s/super-frame)	0.0 – 0.0
`RIGIDITY`	Set min./max. value of cut-off rigidity (GeV/c)	0.0 – 0.0
`SUNH`	Set min./max. value of Sun elevation (degree)	0.0 – 0.0
`ELEV`	Set min./max. value of target elevation (degree)	0.0 – 0.0
`DYE`	Set min./max. value of target elevation from day earth (degree)	0.0 – 0.0
`NTE`	Set min./max. value of target elevation from night earth (degree)	0.0 – 0.0
`PSEUDO_INTERVAL`	Sets time interval of pseudo events[d]	1.0
`EXIT`	Exit from `MODIFY_PARAM`	—

---

[a] When ACS maneuver flag is OFF

[b] When GIS ON, CPU running, RBM flag OFF, and nominall value of HV and fine gain

[c] When PH mode, nominal bit assignment (10-8-8-5-0), nominal RT window, position method FLF, etc.

[d] No pseudo event is generated if set to 0.0.

Table 6.5: Commands in `MODIFY_PARAM` of `GISREAD`

BNK keyword name	Type	Comment
`GISREAD:FILENAME`	Character*(*)	FRF file name reading now
`GISREAD:PSEUDO_INTERVAL`	Real*8	Pseudo event interval (sec). If 0.0, no pseudo events are generated
`GISREAD:ARRIVAL_TIME`	Real*8	Equal to `ANL:TIME` if timing bits are assigned, otherwise 0.0
`GISREAD:TELEMETRY_TIME`	Real*8	Event time calculated from the word position of the event appeared in telemetry. If no timing bits are assigned, equal to `ANL:TIME`

Table 6.6: Special BNK keywords of `GISREAD`

Command	Function	Default
`SHOW`	Display current settings	—
`ENABLE`	Enable various functions of `GISTUNE`	—
`DISABLE`	Disable various functions of `GISTUNE`	—
`DIRECTORY`	Set default directory name of calibration files	/asca/caldb
`PHTUNE`	Set file names of gain maps	'phfl_s2_v3.table' for S2 'phfl_s3_v3.table' for S3
`HKTUNE`	Enable/disable correction of gain with temperature	ENA
`LDHTUNE`	Enable/disable correction of gain with LDHIT count rate	DIS
`RTTUNE`	Set file names of rise time maps	's2xyrtflf.fits' for S2 's3xyrtflf.fits' for S3
`RTMASK`	Set file names of rise time masks	'rtmask' for S2 'rtmask' for S3
`XYLIN`	Set file names of maps for RAW-DET conversion	'xflf_s2.table' for S2 X-coord. 'yflf_s2.table' for S2 Y-coord. 'xflf_s3.table' for S3 X-coord. 'yflf_s3.table' for S3 Y-coord.
`COORD`	Enable/disable attitude correction	ENA
`ASCALINSKY`	Enable/disable to use ASCALIN-compatible sky coordinates	ENA
`ABERRATION`	Enable/disable aberration correction	ENA
`EULER`	Set reference Euler angles[a]	AUTO
`REFJ2000`	Set reference sky position[b]	AUTO
`ROLLANGLE`	Set roll angle	0.0
`GAINHIST`	Set file name of gain history file	'gis_temp2gain.fits'
`PHGAIN`	Set extra factor for PI value	1.0 for S2 1.0 for S3
`TELDEF`	Set file names of teldef files	gis2teldefv2.fits for S2 gis3teldefv2.fits for S3
`EXIT`	Exit from `MODIFY_PARAM`	—

---

[a] Use an attitude of the first event when AUTO

[b] Use Z-axis direction of the first event when AUTO

Table 6.7: Commands in `MODIFY_PARAM` of `GISTUNE`

BNK keyword name	Type	Comment
`GISTUNE:PI:R8`	Real*8	Value of `GIS:PI` before rounding
`GISTUNE:RTI:R8`	Real*8	Value of `GIS:RTI` before rounding
`GISTUNE:X:R8`	Real*8	Value of `GIS:X` before rounding
`GISTUNE:Y:R8`	Real*8	Value of `GIS:Y` before rounding
`GISTUNE:DETX:R8`	Real*8	Value of `GIS:DETX` before rounding
`GISTUNE:DETY:R8`	Real*8	Value of `GIS:DETY` before rounding

Table 6.8: Special BNK keywords of `GISTUNE`

EVS keyword name	Set when
`GISTUNE:BEGIN`	gistune_begin() is called
`GISTUNE:ENTRY`	gistune_ana() is called
`GISTUNE:OK`	gistune_ana() returns ASCA_ANL_OK
`GISTUNE:EULER ERROR`	gistune_ana() cannot get `ANL:EULER` and returns ASCA_ANL_SKIP
`GISTUNE:LDHIT ERROR`	gistune_ana() cannot get LDHIT count rate and returns ASCA_ANL_SKIP
`GISTUNE:RTMASK`	a event is rejected by the rist time mask and returns ASCA_ANL_SKIP

Table 6.9: Special EVS keywords of `GISTUNE`

Chapter 7

Details on FORTRAN Module

Here are described the rules for a FORTRAN module in detail. In the following description, for simplicity, we assume the name of the module of `USER` in this section.

In en explanation of "Arguments" of the subroutines a character in parentheses means a directional characteristic of the argument; "(I)" means "input argument" which is used to pass a value to the subroutine, "(O)" means "output argument" which is used to return a value to the caller, and "(I/O)" means "modification argument" which is used both to pass a value to the subroutine and to return a value to the caller.

You must not change a value of the input argument in any case. Also, you have to set a value to the output argument before returning. In setting a value you are strongly recommended to use a constant in an included file "asca_anl.inc" as described in the following instead of a raw number. You may modify a value of the modification argument if necessary.

Furthermore, you have to satisfy "Requirements" for each subroutine, and are strongly recommended to follow "Recommendation".

Subroutine

`USER_init` —Initialization routine.

Synopsis

```
Subroutine USER_init(status)
Integer status
Include 'Includes.inc'
```

Arguments

`status` (O) Reserved for the future.
Should be `ASCA_ANL_OK` when it returns.

Required

In this subroutine you have to set up a few flags by calling EVSdef.

```
Call EVSdef('USER:BEGIN')
Call EVSdef('USER:ENTRY')
Call EVSdef('USER:OK')
```

The name of these keyword `'USER:BEGIN'`, `'USER:ENTRY'`, `'USER:OK'` are reserved and can not be changed by the user. Also, the you must declare version number of this module 'USER' like this:

```
Call ANL_put_version('USER','version 1.0')
```

The first argument of ANL_put_version is the module name, and second argument is a string (up to 16 characters) containing version number. You can change a format of expressing the version number as you like.

Comments

This subroutine is called once before starting data analysis. `EVSdef` and `BNKdef` (see §3.2.2) are recommended to be in this subroutine.

Subroutine

`USER_com` — Communication routine

Synopsis

```
Subroutine USER_com(status)
Integer status
Include 'Includes.inc'
```

Arguments

`status`	(O)	Reserved for the future. Should be **`ASCA_ANL_OK`** when it returns.

Required

None.

Comments

This subroutine is called in response to the command **`MODIFY_PARAM`** (see also §2.2). Parameters of analysis (*e.g.* sensor ID) should be asked to an analyzer in this subroutine.

Subroutine

`USER_his` — Histogram booking routine

Synopsis

```
Subroutine USER_his(status)
Integer status
Include 'Includes.inc'
```

Arguments

`status`	(O)	Reserved for the future. Should be `ASCA_ANL_OK` when it returns.

Required

None.

Comments

This subroutine is called in response to the command `ANALYZE_DATA` if specified by the command `BOOK_HISTOGRAM` (see also §2.2). All histograms should be booked in this subroutine.

Subroutine

`USER_bgnrun` — Beginning-of-Run routine.

Synopsis

```
Subroutine USER_bgnrun(status)
Integer status
Include 'Includes.inc'
```

Arguments

`status` (O) Controls the flow of data.

Set `ASCA_ANL_OK`,

if the beginning of the run goes good.

Set `ASCA_ANL_QUIT`,

if you want to discard the current data set (or the current file in many case) after this subroutine. After the discard the ASCA_ANL system terminates to process the current data set without calling analysis entries (`*_bgnrun`) of modules downstream in reference to this module.

Required

At the beginning of this subroutine you have to set one of the EVS flag whose keyword is `'USER:BEGIN'`.

```
Call EVSset('USER:BEGIN')
```

Comments

This subroutine is called once just after each data set (*e.g.* data file) is opened. That means this subroutine is called N times in case of analyzing N sets of data, if no module returns `ASCA_ANL_QUIT` as a status of the subroutine `*_bgnrun`. In this subroutine initialization for each data set should be placed.

Subroutine

`USER_ana` — Analysis routine.

Synopsis

```
Subroutine USER_ana(nevent,eventid,status)
Integer nevent, eventid
Integer status
Include 'Includes.inc'
```

Arguments

`nevent` (I) Sequential number of events.

`eventid` (I) Event ID.

`evnentid = ASCA_ANL_OBS`

when the current photon is the observed one.

`evnentid = ASCA_ANL_PSEUDO`

when the current photon is the pseudo event.

`status` (O) Controls the flow of data.

Classic control values (NOT additive)

Set `ASCA_ANL_OK`,

if the analysis goes good.

Set `ASCA_ANL_QUIT`,

if you want to discard the current data set (or the current file in many case) after this subroutine. After the discard the ASCA_ANL system terminates to process the current data set without calling analysis entries (`*_ana`) of modules downstream in reference to this module. Note that NONE of EVS flags are accumulated for the current photon when this value is set.

Set `ASCA_ANL_SKIP`,

if you want to discard the current photon after this subroutine. After the discard the ASCA_ANL system starts to process the next photon data, skipping analysis entries (`*_ana`) of modules downstream in reference to this module.

Additive control values

Add `ANL_DISCARD`,

if you want to discard the current photon after this subroutine. After the discard the ASCA_ANL system starts to process the next photon data, skipping analysis entries (`*_ana`) of modules downstream in reference to this module.

Add `ANL_ENDLOOP`,

if you want to discard the current data set (or the current file in many case) after this subroutine. After the discard the ASCA_ANL system terminates to process the current data set without calling analysis entries (`*_ana`) of modules downstream in reference to this module.

Add `ANL_NOCOUNT`,

if you DON'T want to accumulate the current EVS flags after this subroutine. This flag DOES NOT affect the procedure to call modules downstream in reference to this module. Note that NONE of EVS flags are accumulated for the current photon when this flag is issued.

Required

At the beginning of this subroutine you have to set one of the EVS flag whose keyword is `'USER:ENTRY'`.

```
Call EVSset('USER:ENTRY')
```

If the data is successfully analyzed and all is good at the end of this subroutine, you must set one of the EVS flag whose keyword is `'USER:OK'`.

```
Call EVSset('USER:OK')
```

Comments

This subroutine is called for each photon in data set. That means this subroutine is called N times in case of analyzing N photons. All you have to write in this subroutine is an analysis for one photon, *e.g.*, to add one photon to an appropriate bin in energy histogram. Iterative call of this subroutine realizes the photon-by-photon analysis.

Subroutine

`USER_endrun` — End-of-Run routine.

Synopsis

```
Subroutine USER_endrun(status)
Integer status
Include 'Includes.inc'
```

Arguments

`status`	(O)	Reserved for the future. Should be `ASCA_ANL_OK` when it returns.

Required

None.

Comments

This subroutine is called once just after an analysis of each data set, if the subroutine `*_bgnrun` of the same module was called before the analysis of the same data set. That means this subroutine is called N times in case of analyzing N sets of data, if no module returns `ASCA_ANL_QUIT` as a status of the subroutine `*_bgnrun`.

Subroutine

`USER_exit` — Exit routine.

Synopsis

```
Subroutine USER_exit(status)
Integer status
Include 'Includes.inc'
```

Arguments

`status`	(O)	Reserved for the future. Should be `ASCA_ANL_OK` when it returns.

Required

None.

Comments

This subroutine is called once just before exiting the ASCA_ANL. You do not have to save histograms you created in your analysis because the histograms are saved in response to the command `SAVE_HISTOGRAM` (see also §2.2) or in response to answering `y` when you are asked by ASCA_ANL, `Save histogram (Y/N): Y ?`, before your exit.

Chapter 8

Details on C Module

Here are described the rules for a C module in detail. In the following description, for simplicity, we assume the name of the module of `User`.

You need to include several header files for a C modules. Here is a list of headers required.

header	belongs to	includes
cfortran.h	CERN library	macros to interface C and FORTRAN
hbook.h	CERN library	macros of cfortran for HBOOK
asca_anl.h	ASCA_ANL	constants for flags
cli.h	ASCA_ANL	macros of cfortran for CLI
com.h	ASCA_ANL	macros of cfortran for COM
bnk.h	ASCA_ANL	macros of cfortran for BNK
evs.h	ASCA_ANL	macros of cfortran for EVS
anl_misc.h	ASCA_ANL	macros of cfortran for utility routines

Also, you have to notice the case of letters here in following points.

- All subroutines belonging to the same module have to have the same case of letters. For example, if you define the name as `User`, you have to name `User_init` and so on, and you can't write it as `USER_init` nor `USer_init`.
- All subroutines belonging to libraries[1] written in FORTRAN must call with a name in upper case letters. For example, the subroutine `BNKdef`[2] should be called as `BNKDEF("User:BEGIN")`.

In en explanation of "Arguments" of the subroutines a character in parentheses means a directional characteristic of the argument; "(I)" means "input argument" which is used to pass a value to the subroutine, "(O)" means "output argument" which is used to return a value to the caller, and "(I/O)" means "modification argument" which is used both to pass a value to the subroutine and to return a value to the caller.

You must not change a value of the input argument in any case. Also, you have to set a value to the output argument before returning. In setting a value you are strongly recommended to use a constant in a header file "asca_anl.h" as described in the following instead of a raw number. You may modify a value of the modification argument if necessary.

Furthermore, you have to satisfy "Requirements" for each subroutine, and are strongly recommended to follow "Recommendation".

---

[1] For example, HBOOK, BNK, EVS, and utility routines of the ASCA_ANL(in libMISC.a) are written in FORTRAN.

[2] The subroutine in BNK defining a keyword.

Subroutine

`User_init` —Initialization routine.

Synopsis

`void User_init(int *status)`

Arguments

`status` (O) Reserved for the future.
Should be `ASCA_ANL_OK` when it returns.

Required

In this subroutine you have to set up a few flags by calling EVSdef.

```
EVSDEF("User:BEGIN")
EVSDEF("User:ENTRY")
EVSDEF("User:OK")
```

The name of these keyword `"User:BEGIN"`, `"User:ENTRY"`, `"User:OK"` are reserved and can not be changed by the user. Also, the you must declare version number of this module "User" like this:

```
ANL_PUT_VERSION("User","version 1.0")
```

The first argument of `ANL_PUT_VERSION` is the module name, and second argument is a string (up to 16 characters) containing version number. You can change a format of expressing the version number as you like.

Comments

This subroutine is called once before starting data analysis. `EVSDEF` and `BNKDEF` (see §3.2.2) are recommended to be in this subroutine.

Subroutine

`User_com` — Communication routine

Synopsis

`void User_com(int *status)`

Arguments

`status`	(O)	Reserved for the future. Should be **`ASCA_ANL_OK`** when it returns.

Required

None.

Comments

This subroutine is called in response to the command **`MODIFY_PARAM`** (see also §2.2). Parameters of analysis (*e.g.* sensor ID) should be asked to an analyzer in this subroutine.

Subroutine

`User_his` — Histogram booking routine

Synopsis

`void User_his(int *status)`

Arguments

`status`	(O)	Reserved for the future. Should be `ASCA_ANL_OK` when it returns.

Required

None.

Comments

This subroutine is called in response to the command `ANALYZE_DATA` if specified by the command `BOOK_HISTOGRAM` (see also §2.2). All histograms should be booked in this subroutine.

Subroutine

`User_bgnrun` — Beginning-of-Run routine.

Synopsis

`void User_bgnrun(int *status)`

Arguments

`status` (O) Controls the flow of data.

Set `ASCA_ANL_OK`,

if the beginning of the run goes good.

Set `ASCA_ANL_QUIT`,

if you want to discard the current data set (or the current file in many case) after this subroutine. After the discard the ASCA_ANL system terminates to process the current data set without calling analysis entries (`*_bgnrun`) of modules downstream in reference to this module.

Required

At the beginning of this subroutine you have to set one of the EVS flag whose keyword is `"User:BEGIN"`.

```
EVSSET("User:BEGIN")
```

Comments

This subroutine is called once just after each data set (*e.g.* data file) is opened. That means this subroutine is called N times in case of analyzing N sets of data, if no module returns `ASCA_ANL_QUIT` as a status of the subroutine `*_bgnrun`. In this subroutine initialization for each data set should be placed.

Subroutine

`User_ana` — Analysis routine.

Synopsis

`void User_ana(int nevent,int eventid,int *status)`

Arguments

`nevent` (I) Sequential number of events.

`eventid` (I) Event ID.

`evnentid = ASCA_ANL_OBS`
when the current photon is the observed one.

`evnentid = ASCA_ANL_PSEUDO`
when the current photon is the pseudo event.

`status` (O) Controls the flow of data.

Classic control values (NOT additive)

Set `ASCA_ANL_OK`,
if the analysis goes good.

Set `ASCA_ANL_QUIT`,
if you want to discard the current data set (or the current file in many case) after this subroutine. After the discard the ASCA_ANL system terminates to process the current data set without calling analysis entries (`*_ana`) of modules downstream in reference to this module. Note that NONE of EVS flags are accumulated for the current photon when this value is set.

Set `ASCA_ANL_SKIP`,
if you want to discard the current photon after this subroutine. After the discard the ASCA_ANL system starts to process the next photon data, skipping analysis entries (`*_ana`) of modules downstream in reference to this module.

Additive control values

Add `ANL_DISCARD`,
if you want to discard the current photon after this subroutine. After the discard the ASCA_ANL system starts to process the next photon data, skipping analysis entries (`*_ana`) of modules downstream in reference to this module.

Add `ANL_ENDLOOP`,
if you want to discard the current data set (or the current file in many case) after this subroutine. After the discard the ASCA_ANL system terminates to process the current data set without calling analysis entries (`*_ana`) of modules downstream in reference to this module.

Add `ANL_NOCOUNT`,
if you DON'T want to accumulate the current EVS flags after this subroutine. This flag DOES NOT affect the procedure to call modules downstream in reference to this module. Note that NONE of EVS flags are accumulated for the current photon when this flag is issued.

Required

At the beginning of this subroutine you have to set one of the EVS flag whose keyword is `"User:ENTRY"`.

```
EVSSET("User:ENTRY")
```

If the data is successfully analyzed and all is good at the end of this subroutine, you must set one of the EVS flag whose keyword is `"User:OK"`.

```
EVSSET("User:OK")
```

Comments

This subroutine is called for each photon in data set. That means this subroutine is called N times in case of analyzing N photons. All you have to write in this subroutine is an analysis for one photon, *e.g.*, to add one photon to an appropriate bin in energy histogram. Iterative call of this subroutine realizes the photon-by-photon analysis.

Subroutine

`User_endrun` — End-of-Run routine.

Synopsis

`void User_endrun(int *status)`

Arguments

`status`	(O)	Reserved for the future. Should be `ASCA_ANL_OK` when it returns.

Required

None.

Comments

This subroutine is called once just after an analysis of each data set, if the subroutine `*_bgnrun` of the same module was called before the analysis of the same data set. That means this subroutine is called N times in case of analyzing N sets of data, if no module returns `ASCA_ANL_QUIT` as a status of the subroutine `*_bgnrun`.

Subroutine

`User_exit` — Exit routine.

Synopsis

`void User_exit(int *status)`

Arguments

`status`	(O)	Reserved for the future. Should be **`ASCA_ANL_OK`** when it returns.

Required

None.

Comments

This subroutine is called once just before exiting the ASCA_ANL. You do not have to save histograms you created in your analysis because the histograms are saved in response to the command **`SAVE_HISTOGRAM`** (see also §2.2) or in response to answering **`y`** when you are asked by ASCA_ANL, `Save histogram (Y/N): Y ?`, before your exit.

Appendix A

Installation of the ASCA_ANL

You can install the ASCA_ANL easily in a following way.

1. Install FTOOLS, CERN libraries, etc.
2. Get a copy of source files with ftp.
3. Expand the sources on your computer system.
4. Modify Makefile for your computer system.
5. Make the ASCA_ANL.
6. Install the ASCA_ANL.
7. Install Test.

A.1 Install FTOOLS, CERN libraries, etc.

You must install following packages before building ASCA_ANL.

FTOOLS (includes FITSIO, CFITSIO)
`ftp://legacy.gsfc.nasa.gov/software/ftools/release/`

CERN libraries (includes hbook.h, cfortran.h)
`ftp://asis01.cern.ch/`

atFunctions
`ftp://ftp.astro.isas.ac.jp/astro-d/planning/program/`

COM + CLI [+ readline] (from DISPLAY45)
`ftp://ftp.astro.isas.ac.jp/pub/display/` (DISPLAY45)
`ftp://utsun.s.u-tokyo.ac.jp/ftpsync/prep/` (readline)

ascatool
`ftp://ftp.astro.isas.ac.jp/astro-d/planning/program/`

A.2 FTP

The distribution kit of the ASCA_ANL is placed on a site[1] which is announced with a release note. The kit has a name like `asca_anl_v2.00.tar.gz` which means "ASCA_ANL version 2.00". Transfer the kit to your computer system with ftp.

In the following part of this section,for simplicity, we assume the version number of the ASCA_ANL system as 2.00.

A.3 Expand

After you copy the kit to an appropriate directory on your computer system, type

```
% gzcat asca_anl_v2.00.tar.gz | tar xvf -
```

A directory `asca_anl_v2.00/` are created in the directory where you invoke the command.

The directory structure of below "`asca_anl_v2.00/`" are following:

`asca_anl_v2.00/`

`src/`	··· contains source files
`module/`	··· contains modules
`include/`	··· contains header files
`util/`	··· contains utilities
`tools/`	··· contains tools
`doc/`	··· contains documentations
`sample/`	··· contains sample modules

A.4 Arrange Makefile

Edit a file

```
asca_anl_v2.00/Includes.make
```

and change definitions of the following macro to specify a directory.

A.5 Make

In order to "make" the ASCA_ANL, just type

```
% make
```

in a directory `asca_anl_v2.00/`. It takes about 2 minutes to compile whole package on DEC 3000/600S AXP.

A.6 Install

Create install directories where you wrote in "`Includes.make`" (`ASCA_ANL_INC`, `ASCA_ANL_LIB`, `ASCA_ANL_BIN`), and type '`make install`'.

After installing ASCA_ANL package, you can erase object files and libraries by typing '`make clean`'.

---

[1] The site is not an anonymous ftp site. Thus you are required to have your own account on the site.

`MACRO`	required contents in the directory
`ASCA_ANL_DIR`	the ASCA_ANL directory
`ASCA_ANL_INC`	header files of the ASCA_ANL
`ASCA_ANL_LIB`	libraries of the ASCA_ANL
`ASCA_ANL_BIN`	executables for the ASCA_ANL
`ASCA_ANL_LNK`	link option for the ASCA_ANL
`ASCATOOL_DIR`	ascatool directory
`ASCATOOL_INC`	header files for ascatool `ascatool.h`, etc.
`ASCATOOL_LNK`	link option for ascatool `libascatool.a`
`COM_CLI_LNK`	link option for CLI/COM `libCOM.a`, `libCLI.a`, `libreadline.a`
`ATFUNCTIONS_INC`	header files for atFunctions `atFunctions.h` and `atError.h`
`ATFUNCTIONS_LNK`	link option for the atFunctions `libatFunctions.a`
`CERN_DIR`	CERN directory
`CERN_INC`	header files for CERN library `cfortran.h` and `hbook.h`
`CERN_LNK`	link option for CERN library `libgraflib.a`, `libgrafX11.a`, `libpacklib.a`, `libkernlib.a`, `libmathlib.a`, `libX11.a`
`FITSIO_LNK`	link option for FITSIO `libfitsio.a`
`CC`	default C compiler
`FC`	default FORTRAN compiler
`DEFAULT_F77_LNK`	default FORTRAN library

Table A.1: Macros in `Include.make`

A.7 Install Test

You are recommended to check your installation of ASCA_ANL.
Type as follows:

```
% cd sample
% make
% make test
```

If something is wrong, '`make`' or '`make test`' complains error messages. Read those messages and check "Includes.make" again. After test, you shuld clean sample directories for ASCA_ANL users, typing '`make clean`' in "`./sample/`" directory.

Appendix B

BNK

A following description shows a typical procedure to use BNK. See TableB.1 in details.

B.1 In FORTRAN

1. First, declare "BNK keyword"[1] and size of data.

   ```
   Call BNKdef(key, size)
   ```

   Input parameter

`key`	BNK keyword to be defined	(Character $\leq$ 24chars)
`size`	Size of stored data in byte	(Integer $>$ 0)

2. Put data into BNK

   ```
   Call BNKput(key, size, array)
   ```

   Input parameter

`key`	BNK keyword	(Character $\leq$ 24chars)
`size`	Size of data in byte	(Integer $>$ 0)
`array`	Variable storing data to put	(type of data to be put)

3. Get data from BNK

   ```
   Call BNKget(key, size, used_size, array)
   ```

   Input parameter

`key`	BNK keyword	(Character $\leq$ 24chars)
`size`	Size of data to get in byte	(Integer $>$ 0)

   Output parameter

`used_size`	Size of stored data in byte	(Integer $>$ 0)
`array`	Variable to get data	(type of data to be got)

---

[1] "BNK keyword" is the name to be used to access each data.

Subroutine name	function
`BNKini`	initialize
`BNKdef(key,size)`	BNK key definition
`BNKeqv(key1,size,key,start_pointer)`	assign key to predefined (key and data)
`BNKput(key,size,array)`	put data
`BNKget(key,size,used_size,array)`	get data
`BNKlst`	list key
`BNKdmp(key,format)`	dump data to STDOUT
`BNKnum(key,stored_size,maxsize)`	get information
`BNKclr`	clear of data

Table B.1: Subroutines of BNK

B.2 In C

1. First, declare “BNK keyword”[2] and size of data.

   `BNKDEF(key, size)`

   Input parameter

`key`	BNK keyword to be defined	(char $\leq$ 24chars)
`size`	Size of stored data in byte	(int $>$ 0)

2. Put data into BNK

   `BNKPUT(key, size, ptr)`

   Input parameter

`key`	BNK keyword	(char $\leq$ 24chars)
`size`	Size of data in byte	(int $>$ 0)
`ptr`	Pointer to variable storing data to put	(pointer)

3. Get data from BNK

   `BNKGET(key, size, used_size, ptr)`

   Input parameter

`key`	BNK keyword	(char $\leq$ 24chars)
`size`	Size of data to get in byte	(int $>$ 0)

   Output parameter

`used_size`	Size of stored data in byte	(int $>$ 0)
`ptr`	Pointer to variable to get data	(pointer)

---

[2] “BNK keyword” is the name to be used to access each data.

Appendix C

EVS

1. First, declare the name of event selection.

   ```
   Call EVSDEF( name )
   ```

   will define the name (charcter string within 32 chars).

2. There are three ways to set the value of event selection.

   (a) EVSSET(name)

   ```
   Call EVSSET( name )
   ```

   will set the value of the specified event selection to .TRUE. .

   (b) EVSCLR(name)

   ```
   Call EVSCLR( name )
   ```

   will set the value of the name to .FALSE. ;in other word, 'clear'.

   (c) EVSVAL(name, logic)

   ```
   Call EVSVAL( name, logic )
   ```

   will set the value of the event selection to the value given by 'logic'. 'logic' is a logical expression, as .TRUE., .FALSE., logical variable, or logical combination of logical variables.

3. In order to get the value of the event selection, use logical function EVS.

   ```
   EVS( name )
   ```

   returns the value of the specified event selection. For example, you can use EVS as

   ```
    If( EVS( 'ALL_CHAMBERS_HIT' ) ) then
      ......
    or
    Call EVSVAL( 'PC1V_IS_OK', ( MULT(1).GT.0 ) )
    Call EVSVAL( 'PC1H_IS_OK', ( MULT(2).GT.0 ) )
    Call EVSVAL( 'PC1_IS_OK',
   &             ( EVS('PC1V_IS_OK') .and. EVS('PC1H_IS_OK') ) )
    ...
   ```

   and so on. Do not forget to declare the function EVS as logical ; put the next statement in a declaration part in each routine which uses EVS.

Appendix D

HBOOK

To use HBOOK, you have to know only two concepts at least, *i.e.*, booking histograms and filling data to them. Histograms can be one dimensional or two dimensional. HBOOK supports another type of histograms and various useful functions to operate them. See *HBOOK Reference Manual* [1] in details.

We hope following descriptions will helpful for you.

1. Book Histogram
   — One dimensional case —

```
Call HBOOK1 (ID, TITLE, NX, XMIN, XMAX, 0.0)
```

Input parameter

`ID`	Histogram Identifer	(INTEGER > 0)
`TITLE`	Histogram Title	(CHARACTER)
`NX`	Number of channels	(INTEGER)
`XMIN`	Lower edge of first channel	(REAL)
`XMAX`	Upper edge of last channel	(REAL)

— two dimensional case —

```
Call HBOOK2 (ID, TITLE, NX, XMIN, XMAX, NY, YMIN, YMAX, 0.0)
```

Input parameter

`ID`	Histogram Identifer	(INTEGER > 0)
`TITLE`	Histogram Title	(CHARACTER)
`NX`	Number of channels in X	(INTEGER)
`XMIN`	Lower edge of first X channel	(REAL)
`XMAX`	Upper edge of last X channel	(REAL)
`NY`	Number of channels in Y	(INTEGER)
`YMIN`	Lower edge of first Y channel	(REAL)
`YMAX`	Upper edge of last Y channel	(REAL)

2. Fill Data into Histogram
   — One dimensional case —

```
Call HF1 (ID, X, WEIGHT)
```

Input parameter

`ID`	Histogram Identifer	(INTEGER > 0)
`X`	value of the abscissa	(REAL)
`WEIGHT`	event weight	(REAL)

---

[1] *HBOOK Reference Manual* is distributed with HBOOK by CERN.

— two dimensional case —

`Call HF2 (ID, X, Y, WEIGHT)`

Input parameter

`ID`	Histogram Identifer	(INTEGER > 0)
`X`	value of the abscissa	(REAL)
`Y`	value of the ordinate	(REAL)
`WEIGHT`	event weight	(REAL)

Appendix E

CLI Special Directives

`@file [arg...]`

Executes a command procedure file '`file`'. '`%n`' (n=1..9) in a command procedure file will be replaced by the actual arguments.

`@@file`

Executes a command procedure file 'file' in the single step mode.

`@*prompt`

Reads from TT in a command procedure.

`@<`

Suspends/resumes the execution of a command procedure.

`@!comment`

Comment in a command procedure.

`@$command`

Executes a DCL/shell command.

`@(file`

The subsequent inputs will be written on 'file'.

`@)`

Closes the file opened by '`@(`' directive.

`@>file`

The subsequent outputs will be written on 'file'.

`@>*`

Closes the file opened by '`@>`' directive and resumes the output.

`@%`

Turns outputs off.

`@#`

Turns outputs on.

`@symbol=value`

Symbol assignment. The symbol will be referred as `$symbol` or `$(symbol)`.

`!!`

Recalls and executes a most recently entered command line.

`!-`*n*

Recalls and executes a *n*-th recently entered command line.

`!`*pattern*

Recalls and executes a most recently entered command line that matches the characters '*pattern*'.

Bibliography

[1] T.Takahashi, Mini Data Bank system —BNK—

[2] T.Takahashi Event Selection Software —EVS—

[3] HBOOK Reference Manual Version 4.15,
CERN Program Library Long Writeup Y250,
Application Software Group, Computing and Networks Division,
CERN Geneva, Switzerland

[4] HPLOT user guide,
R.Brun and N. Cremel Somon,
CERN program library Y251, February 19, 1988

[5] MINUIT user guide,
F.James and M.Ross,
CERN program library D506, December, 1989

[6] DISPLAY,
B. Gabioud, H.L.Videau, M.A.Garnjost, W.Koellner,
E.Wang, T.Takahashi, H.Yamamoto, R.Avery,
Group A Physics Programming Note P-269, TPC Note No. TPC-LBL-80-37

[7] DISPLAY45,
A. Sirahashi, T. Takahashi and R. Itoh
Dept. of Physics, Univ. of Tokyo and
KEK, 1993
(miranda:)/utk/lib/doc/display45.ps

[8] PEP-4 Command Interpreter,
B.Gaioud,
Group A Physics Programming Note P-278, TPC Note No. TPC-LBL-82-15

[9] CFORTRAN 2.2 : Interfacing C and FORTRAN,
Burkhard Burow, U. of Toronto, 1991
in a text file cfortran.doc, available via anonymous ftp at zebra.desy.de (131.169.2.244).