M212927EN-D

Programming Guide

IRIS Radar

IRIS



VAISALA

PUBLISHED BY

Vaisala Oyj
Vanha Nurmijärventie 21, FI-01670 Vantaa, Finland
P.O. Box 26, FI-00421 Helsinki, Finland
+358 9 8949 1
vaisala.com
docs.vaisala.com

© Vaisala 2026

No part of this document may be reproduced, published, or publicly displayed in any form or by any means, electronic or mechanical (including photocopying), nor may its contents be modified, translated, adapted, sold, or disclosed to a third party without prior written permission of the copyright holder. Translated document and translated portions of the document are based on the original English version. Parts of the translation may have been created using machine-assisted translation. In ambiguous cases, the English version is applicable, not the translation.

The contents of this document are subject to change without prior notice.

Local rules and regulations applicable to the products and services may vary, and they shall take precedence over the information contained in this document. Vaisala makes no representations on this document's compliance with the local rules and regulations applicable at any given time, and hereby disclaims any and all responsibilities related thereto. You are instructed to confirm the applicability of the local rules and regulations and their effect on the intended use of the products and services.

This document does not create any legally binding obligations for Vaisala towards customers or end users. All legally binding obligations are set forth exclusively in the applicable contract or in the relevant set of General Conditions of Vaisala (vaisala.com/policies).

This product contains software developed by Vaisala or third parties. Use of the software is governed by license terms and conditions included in the applicable contract or, in the absence of separate license terms and conditions, by the General License Conditions of Vaisala Group.

This product may contain open-source software (OSS) components. In the event this product contains OSS components, then such OSS is governed by the terms and conditions of the applicable OSS licenses, and you are bound by the terms and conditions of such licenses in connection with your use and distribution of the OSS in this product. Applicable OSS licenses are included in the product itself or provided to you on any other applicable media, depending on each individual product and the product items delivered to you.

Table of contents

1. About this document 9
1.1 Version information 9
1.2 Related documents 10
1.3 Documentation conventions 10
1.4 Trademarks 11

2. Introduction to IRIS programming 12
2.1 IRIS programming interface 12
2.2 Directory organization 12
2.2.1 File system hierarchy tree 13
2.3 Setting up the development environment 14
2.3.1 Setting-up a developer source tree 14
2.3.2 Compiling and linking an IRIS library or application 15
2.3.3 Compiling IRIS HDF5 code 15
2.3.4 Compiling IRIS BUFR pipes 16

3. Custom input and output 17
3.1 Input pipes 17
3.2 User Product Insert 18
3.3 Building example programs 19
3.4 Running the upi_rcv program 19
3.5 IRIS product and data types 20
3.6 Customizing the legend 21

4. Data formats 24
4.1 Data format overview 24
4.2 Scalar definitions 24
4.3 Structure definitions 25
4.3.1 beam_psi_struct structure 25
4.3.2 cappi_psi_struct structure 25
4.3.3 catch_psi_struct structure 26
4.3.4 catch_results structure 26
4.3.5 color_scale_def structure 27
4.3.6 cross_psi_struct structure 28
4.3.7 dsp_data_mask structure 29
4.3.8 extended_header_v0 structure 29
4.3.9 extended_header_v1 structure 29
4.3.10 extended_header_v2 structure 30
4.3.11 fcast_psi_struct structure 30
4.3.12 gage_psi_struct structure 31
4.3.13 gage_results structure 31
4.3.14 ingest_configuration structure 32
4.3.15 ingest_data_header structure 33
4.3.16 ingest_header structure 34
4.3.17 max_psi_struct structure 34
4.3.18 mlhgt_psi_struct structure 35
4.3.19 ndop_input structure 35
4.3.20 ndop_psi_struct structure 35
4.3.21 ndop_results structure 36
4.3.22 one_protected_region structure 36
4.3.23 ppi_psi_struct structure 37
4.3.24 product_configuration structure 37

4.3.25 product_end structure..... 40
4.3.26 product_hdr structure..... 43
4.3.27 product_header_extensions..... 44
4.3.28 product_specific_info structure..... 44
4.3.29 protect_setup structure..... 45
4.3.30 rain_psi_struct structure..... 45
4.3.31 raw_prod_bhdr structure..... 46
4.3.32 raw_psi_struct structure..... 46
4.3.33 ray_header structure..... 47
4.3.34 rhi_psi_struct structure..... 47
4.3.35 rti_psi_struct structure..... 47
4.3.36 shear_psi_struct structure..... 48
4.3.37 sline_psi_struct structure..... 48
4.3.38 sline_results structure..... 49
4.3.39 sri_psi_struct structure..... 51
4.3.40 status_antenna_info structure..... 52
4.3.41 status_device_info structure..... 53
4.3.42 status_message_info structure..... 53
4.3.43 status_misc_info structure..... 53
4.3.44 status_one_device structure..... 54
4.3.45 status_one_process structure..... 54
4.3.46 status_process_info structure..... 55
4.3.47 status_results structure..... 55
4.3.48 structure_header structure..... 55
4.3.49 tape_header_record structure..... 56
4.3.50 task_calib_info structure..... 56
4.3.51 task_configuration structure..... 58
4.3.52 task_dsp_info structure..... 58
4.3.53 task_dsp_mode_batch structure..... 59
4.3.54 task_end_info structure..... 60
4.3.55 task_file_scan_info structure..... 60
4.3.56 task_manual_scan_info structure..... 61
4.3.57 task_misc_info structure..... 61
4.3.58 task_ppi_scan_info structure..... 61
4.3.59 task_range_info structure..... 62
4.3.60 task_rhi_scan_info structure..... 62
4.3.61 task_scan_info structure..... 62
4.3.62 task_sched_info structure..... 63
4.3.63 tdwr_psi_struct structure..... 64
4.3.64 tdwr_results structure..... 64
4.3.65 text_results structure..... 65
4.3.66 top_psi_struct structure..... 65
4.3.67 track_psi_struct structure..... 65
4.3.68 track_results structure..... 66
4.3.69 vad_psi_struct structure..... 67
4.3.70 vad_results structure..... 67
4.3.71 vvp_psi_struct structure..... 67
4.3.72 vvp_results structure..... 68
4.3.73 warn_psi_struct structure..... 69
4.3.74 warning_results structure..... 69
4.3.75 wind_psi_struct structure..... 70
4.3.76 wind_results structure..... 70
4.3.77 ymds_time structure..... 71

4.4 Data types....71
4.4.1 Extended header format (DB_XHDR)....73
4.4.2 2-byte axis of dilation format (DB_AXDIL2)....73
4.4.3 1-byte AZDR format (DB_AZDR8)....74
4.4.4 2-byte AZDR format (DB_AZDR16)....74
4.4.5 1-byte Clutter Signal Power format (DB_CSP8)....74
4.4.6 2-byte Clutter Signal Power (DB_CSP16)....74
4.4.7 1-byte CSR format (DB_CSR8)....74
4.4.8 2-byte CSR format (DB_CSR16)....74
4.4.9 2-byte deformation format (DB_DEFORM2)....74
4.4.10 2-byte divergence format (DB_DIVERGE2)....75
4.4.11 1-byte echo tops format (DB_HEIGHT)....75
4.4.12 2-byte floating liquid format (DB_FLIQUID2)....75
4.4.13 2-byte horizontal wind direction format (DB_HDIR2)....76
4.4.14 1-byte HydroClass format (DB_HCLASS)....76
4.4.15 2-byte HydroClass format (DB_HCLASS2)....78
4.4.16 1-byte integral attenuation (DB_AH8) and (DB_AV8)....78
4.4.17 2-byte integral attenuation (DB_AH16) and (DB_AV16)....79
4.4.18 1-byte KDP format (DB_KDP)....79
4.4.19 2-byte KDP format (DB_KDP2)....80
4.4.20 1-byte LDR format (DB_LDRH & DB_LDRV)....80
4.4.21 2-byte LDR format (DB_LDRH2 & DB_LDRV2)....81
4.4.22 1-byte LOG format (DB_LOG8)....81
4.4.23 2-byte LOG format (DB_LOG16)....81
4.4.24 1-byte Phi format (DB_PHIH & DB_PHIV)....81
4.4.25 2-byte Phi format (DB_PHIH2 & DB_PHIV2)....81
4.4.26 1-byte PhiDP format (DB_PHIDP)....81
4.4.27 2-byte PhiDP format (DB_PHIDP2)....82
4.4.28 1-byte PMI format (DB_PMI8)....82
4.4.29 2-byte PMI format (DB_PMI16)....82
4.4.30 2-byte Rainfall rate format (DB_RAINRATE2)....82
4.4.31 1-byte reflectivity format (DB_DBT& DB_DBZ)....83
4.4.32 2-byte reflectivity format (DB_DBT2& DB_DBZ2)....84
4.4.33 1-byte Rho format (DB_RHOH & DB_RHOV)....84
4.4.34 2-byte Rho format (DB_RHOH2 & DB_RHOV2)....84
4.4.35 1-byte RhoHV format (DB_RHOHV)....84
4.4.36 2-byte RhoHV format (DB_RHOHV2)....85
4.4.37 1-byte SNR format (DB_SNR8)....85
4.4.38 2-byte SNR format (DB_SNR16)....85
4.4.39 1-byte Signal Quality Index format (DB_SQI)....85
4.4.40 2-byte Signal Quality Index format (DB_SQI2)....86
4.4.41 2-byte time format (DB_TIME2)....86
4.4.42 1-byte unfolded velocity format (DB_VELC)....86
4.4.43 2-byte unfolded velocity format (DB_VELC2)....87
4.4.44 1-byte velocity format (DB_VEL)....87
4.4.45 2-byte velocity format (DB_VEL2)....88
4.4.46 2-byte vertical velocity format (DB_VVEL2)....88
4.4.47 2-byte VIL format (DB_VIL2)....88
4.4.48 1-byte width format (DB_WIDTH)....89
4.4.49 2-byte width format (DB_WIDTH2)....89
4.4.50 1-byte wind shear format (DB_SHEAR)....89

4.4.51 1-byte XCOR format (DB_XCOR8)..... 90
4.4.52 2-byte XCOR format (DB_XCOR8).....90
4.4.53 1-byte ZDR format (DB_ZDR).....90
4.4.54 2-byte ZDR format (DB_ZDR2)..... 91
4.5 Ingest data file format..... 91
4.5.1 Ingest file names.....92
4.6 Product file format..... 92
4.6.1 Product file names.....93
4.6.2 Cartesian product format.....93
4.6.3 FCAST product format.....93
4.6.4 MLHGT product format.....94
4.6.5 NDOP product format.....94
4.6.6 RAW product format.....94
4.6.7 SLINE product format..... 97
4.6.8 TDWR product format..... 97
4.6.9 TRACK product format..... 97
4.6.10 VAD product format.....98
4.6.11 VVP product format.....98
4.6.12 WARN product format.....98
4.6.13 WIND product format.....98
4.7 Tape format..... 99
4.8 TIFF output format.....99
4.9 Constants.....100

5. Information utilities..... 103
5.1 Productx.....103
5.1.1 Invoking productx.....103
5.1.2 Productx examples.....105
5.2 Rays utility.....107
5.2.1 Invoking rays.....107
5.2.2 Headers only example..... 108
5.2.3 Velocity example.....109
5.2.4 Extended header example..... 110

Appendix A: structmap command..... 112

Appendix B: Radar Control Protocol..... 114

Appendix C: Link transmission formats..... 128
C.1 AWS (Austrian Weather Service) format..... 128
C.2 Hong Kong Observatory format.....129

Appendix D: UF format..... 131
D.1 UF format overview.....131
D.2 Single UF ray structure..... 131
D.3 uf_data_header2 structure.....132
D.4 uf_field_header2 structure..... 132
D.5 uf_fsi2 structure.....133
D.6 uf_mandatory_header2 structure..... 133
D.7 uf_optional_header structure..... 135

Appendix E: RTD format..... 136
E.1 Real Time Display overview..... 136
E.2 Rtd_nids3_xmt.....137
E.3 Rtd_v1_xmt.....137

E.4 Rtd_v2_xmt 137

Appendix F: Supported IRIS input pipes 139

Appendix G: Supported IRIS output pipes 141

Appendix H: Common-DTO protobuf pipes 144

Appendix I: Shell-script wrappers for conversion pipes 145

Warranty 147

Technical support 147

Recycling 147

List of tables

Table 1 Documentation versions (English)....9
Table 2 Related documents (English)....10
Table 3 Input pipe message returns....17
Table 4 Example programs....19
Table 5 Product Types....21
Table 6 out_legend.dat flag values....22
Table 7 IRIS data types....24
Table 8 IRIS Timezone Recording....43
Table 9 Data types....71
Table 10 Method 1 - METEOCLASSIFIER (HydroClass)....77
Table 11 Method 2 - PRECIPCLASSIFIER....77
Table 12 Method 3 - CELLCLASSIFIER....78
Table 13 Ingest data file format....91
Table 14 Product file format....92
Table 15 Compression code meanings....96
Table 16 Raw product example....96
Table 17 TIFF fields used by IRIS....99
Table 18 Data type constants — /include/sigtypes.h....100
Table 19 Raw Product Parameters....103
Table 20 Product file naming....103
Table 21 Ray options....108
Table 22 Status packet RCV01 format (RCP to host)....115
Table 23 Control packet XMT01 format (host to RCP)....116
Table 24 Status packet RCV02 / RCV04 format (RCP to host)....117
Table 25 Control packet XMT02 / XMT04 format (host to RCP)....118
Table 26 Status packet RCV03 format (RCP to host)....119
Table 27 Status packet RCV05 format (RCP to host)....122
Table 28 Control packet XMT05 format (host to RCP)....123
Table 29 Time packet (RCP to host)....123
Table 30 Generic BITE status packet (both ways)....124
Table 31 BITE command packet (both ways)....124
Table 32 Auxiliary control BITE packets (both ways)....125
Table 33 Q-BITE Status Packet (Both ways)....126
Table 34 Simple Q-BITE example....126
Table 35 Q-BITE interrogate packet (both ways)....126
Table 36 BITE individual command packet (host to RCP)....127
Table 37 Chat mode packet (both ways)....127
Table 38 HKO picture types....130
Table 39 Network load from RTD dependencies....136
Table 40 Latest input pipes....139
Table 41 Legacy input pipes....140
Table 42 Latest output pipes....141
Table 43 Legacy output pipes....142
Table 44 Common DTO protobuf pipes....144

1. About this document

1.1 Version information

This document provides an introduction to the IRIS programming interface.

Table 1 Documentation versions (English)

Document code	Date	Description
M212927EN-D	June 2026	Updated for IRIS RDA 10.5. Information about supported input and output pipes was updated. See the following chapters: • Input pipes (page 17) • Supported IRIS input pipes (page 139) • Supported IRIS output pipes (page 141) • Common-DTO protobuf pipes (page 144)

M212927EN-C	February 2025	This document version is for IRIS 10.2.0.
		Updates include:
		• The following pipes were updated:`iris2odimhdf5` (Previously named `IrisToOdimHDF5`) and `df52iris`. See Supported IRIS output pipes (page 141) and Supported IRIS input pipes (page 139).
		• The 1-byte and 2-byte SNR range and resolution information was updated:
		  • 1-byte SNR format (DB_SNR8) (page 85)
		  • 2-byte SNR format (DB_SNR16) (page 85)
		• Notification conventions updated according to the ANSI Z535.6 standard. See Documentation conventions (page 10).

M212927EN-B	August 2024	This document version is for IRIS 10.1.0. Updates: • File hierarchy tree updated. • Moved **`structmap`** command information to this document from *IRIS and RDA Software Installation Guide (M212924EN)*.

Document code	Date	Description
M212927EN-A	May 2023	First version of this document. This document describes IRIS v. 10.0.0, which supports RVP10 Digital Receiver and Signal Processor. The predecessor of this document is M211318EN, which describes IRIS 9.1.0 and earlier versions

1.2 Related documents

Table 2 Related documents (English)

Document code	Name
DOC236879	*IRIS RDA Release Notes*
M212924EN	*IRIS and RDA Software Installation Guide*
M212925EN	*IRIS and RDA Utilities Guide*
M212926EN	*IRIS Radar User Guide*
M212927EN	*IRIS Programming Guide*
M212928EN	*IRIS Product and Display Guide*
M212923EN	*Radar Control Processor RCP8 User Guide*
M212604EN	*RVP10 Digital Receiver and Signal Processor User Guide*

1.3 Documentation conventions

NOTICE! Alerts you to a situation that, if not avoided, could result in the product to be damaged, or important data to be lost.

Highlights important information on using the product.

Gives tips for using the product more efficiently.

1.4 Trademarks

Vaisala® and WindCube® are registered trademarks and HydroClass™, IRIS™ and Total Lightning Processor™ are trademarks of Vaisala Oyj.

Google Chrome™ is a trademark of Google Inc.

Mozilla™ and Firefox™ are trademarks of the Mozilla Foundation.

Microsoft Edge® is a trademark of Microsoft Corporation in the United States and other countries.

All other product or company names that may be mentioned in this publication are trade names, trademarks, or registered trademarks of their respective owners.

2. Introduction to IRIS programming

2.1 IRIS programming interface

The IRIS programming interface provides access to the radar, antenna, and signal processing hardware, as well as to the IRIS software.

You can expand the capabilities of IRIS to, for example:

- Create special-purpose products
- Write applications that accept data generated by IRIS
- Write new interfaces to IRIS

2.2 Directory organization

IRIS source files, object modules, and library routines are in separate branches of the IRIS directory tree.

IRIS_ROOT refers to the root directory for the IRIS system. Often `/usr/sigmet`; `IRIS_ROOT` is defined as an environment variable.

In this document, pathnames are relative to *IRIS_ROOT*, unless otherwise noted. All source code is in the */src* directory.

All system wide header files are in the *${IRIS_ROOT}/include* directory, and all library *.a* files are in the *${IRIS_ROOT}/lib* directory.

2.2.1 File system hierarchy tree

```
├── etc
│   ├── vaisala
│   │   └── irisrda
│   │       ├── images
│   │       ├── init
│   │       ├── listings
│   │       ├── menu
│   │       └── overlay
│   │   └── irisrda_templates
│   ├── systemd /* service files */
│   ├── profile.d /* shell script files */
│   ├── rc.d
│   │   └── init.d /* rc files */
│   └── logrotate.d /* logrotate conf files */
├── srv
│   └── iris_data
│       ├── ingest       /* IRIS ingest files */
│       ├── product      /* IRIS derived product files */
│       ├── product_raw  /* IRIS RAW (packed ingest) files */
│       ├── tsarchive    /* saved IQ data */
│       ├── tape_inv     /* tape backup metadata */
│       ├── ascope     /* Data saved from ascope utility */
│       ├── suncal     /* Sun based calibration utility data */
│       ├── temp       /* temp storage */
│       └── zdrcal     /* ZDR calibration utility data */
├── usr
│   ├── libexec
│   │   └── vaisala
│   │       ├── pipes /* pipes from data_converters */
│   │       └── irisrda /* scripts for irisrda */
│   ├── bin /* executables for irisrda */
│   ├── share
│   │    ├── X11 /* app defaults resource files */
│   │    ├── nls /* po files */
│   │    ├── vaisala
│   │    │   └── irisrda
│   │    │       ├── icons /* icon files */
│   │    │       ├── sounds /* sound files */
│   │    │       └── keys /* key files */
│   │    └── doc /* user documentation, release notes and license files */
│   ├── lib64 /* all libraries */
│   └── include
│        └── vaisala/* all header files */
```

```
└── var
    ├── log
    │   └── irisrda /* all log files */
    └── spool
        └── vaisala-io
            ├── input      /* Default IRIS input directory */
            ├── output     /* Default IRIS output directory */
            ├── raw
            ├── volume
            └── temp       /* Conversion pipes temp storage */
```

2.3 Setting up the development environment

You may need to compile your own programs using IRIS headers, which may link to IRIS libraries. Some use cases:

- Writing your own code to read or write IRIS products using the IRIS data structures
- Writing your own input or output pipe program to convert the data to another format
- Writing your own code to interface to the RCP and RVP using our public APIs. For example, for an offline diagnostic program.
- Defining new product types

You must install the source files, object modules, and libraries on your system.

1. Select the corresponding options in the **install** program.
2. See the instructions for your operating system in *IRIS and RDA Software Installation Guide (M212924EN)*.

2.3.1 Setting-up a developer source tree

Before you can write new code, you must set up your working directories.

The main IRIS source code is in the release directory, *`${IRIS_ROOT}/src`*, which includes the major subdirectories: *`libs`*, *`iris`*, and *`utils`*.

1. Copy the files you need and the release *`config.mk`* file.



Do not change the source in the release tree.

2. In the operator account, set up your developer tree with the following commands:

```
$ cd
$ mkdir src
$ cd src
$ cp -r /usr/sigmet/src ./sigmet
```

3. Make and compile your changes.

   If you modify a standard IRIS standard pipes, Vaisala recommends that you rename it with a custom name. That way it does not get replaced by the next upgrade.

2.3.2 Compiling and linking an IRIS library or application

Each directory installed on your system contains a *`Makefile`* to build the application or library it contains.

1. To run the `Makefile`:
   a. Switch to the directory containing the source code.
   b. Type **`make`**.
2. To install the binaries, run: **`sudo make install`**



You can execute **`sudo make install`** in any directory, including root of the tree. However, if you make changes in the *`base`* sub-tree, you must execute **`make`** and **`sudo make install`** before building any binaries in the *`iris`* sub-tree.

3. To compile and install the *`base`* or *`iris`* sub-tree, at the base of the sub-tree type:
   a. **`make`**
   b. **`sudo make install`**



This is usefull when you build or install *`base`* or *`iris`* sub-trees.

2.3.3 Compiling IRIS HDF5 code

You must install the following RPM packages:

- *`hdf5-1.8.5.patch1-5.el6.i686.rpm`*
- *`hdf5-devel-1.8.5.patch1-5.el6.i686.rpm`*

1. If you do not have a release USB stick, retrieve the installation files from: https://ftp.sigmet.com/files/releases
2. Mount the IRIS/RDA release USB stick, and go to the directory *RHEL6/extras/RPMS*.
3. Install with the following command:

```
# rpm -Uhv hdf5-1.8.0-1.el5.rf.i386.rpm
```

2.3.4 Compiling IRIS BUFR pipes

You must install the *bufr* and *bufr-devel* RPM packages.

1. If you do not have a release DVD, retrieve the files from: ftp://ftp.sigmet.com/outgoing/os_patches/RHEL6/RPMS
2. Mount the IRIS/RDA release DVD and go to the directory *RHEL6/extras/RPMS*.
3. As super user, install the packages with the following commands:

```
rpm -Uhv bufr-3.0-2.i386.rpm
rpm -Uhv bufr-devel-3.0-2.i386.rpm
```

3. Custom input and output

3.1 Input pipes

IRIS launches the pipe using one of two command-line formats: **`Pathnames`** or **`Pipe`**.

In chapters Supported IRIS input pipes (page 139) and Supported IRIS output pipes (page 141), the **Syntax** column in the tables indicates which command-line format a pipe expects.

`Pathnames`

IRIS passes the input and output file paths and the pipe opens those files itself:

```
$ <pipe> --ip=<in-path> --op=<out-path> --device=<number>
```

Exception: **`InmToIris`**, **`IrisToAscii`** and **`IrisToEwis`** use the legacy colon spelling instead (**`-ip: -op:`**, **`plus -if`**: to name the input file separately).

`Pipe`

IRIS streams the data through an OS pipe, and the pipe reads from **`stdin`** and/or writes to **`stdout`**. Options, if any, use the colon style, for example: **`-device:`**.

Message returns

The information passed back to the calling program is the 8-bit exit status.

Note:

- Pipes must explicitly open the input and output files.
- The pipe program must write diagnostic information to a log file. Vaisala recommends writing to the *`${IRIS_LOG}`* directory.

Table 3 Input pipe message returns

Message	Description
`EXIT_SUCCESS (0)`	Message returned in normal cases.
`EXIT_FAILURE (1)`	IRIS sends the messages: **`Error running pipe, check log`**.

Message	Description
`EXIT_RERUN (2)`	The pipe has produced a normal output, and needs to produce a second output file but is unable to do so. The pipe must store information about the second output on disk. Vaisala recommends writing this in the *`${IRIS_TEMP}`* directory. IRIS processes the first output, then reinvokes the pipe with the following command line: `$ <pipe-path> -rerun -op:<out-path> -device:<number>` When the pipe is called, it can retrieve the stored information and write the second output to the specified pathname.
`EXIT_NOTHING (3)`	The pipe ran without error, but no output was produced. Useful for a pipe that must read several input files before producing an output file. IRIS ignores and deletes the output file.

3.2 User Product Insert

You can generate and process radar products within their own programs. You can use polling or user product insert (UPI) to pass the product between the user program and IRIS.

Polling

In polling, the product is passed from the user program using IRIS input and output mechanisms with no handshaking.

For example, you can place the product file in a known directory that IRIS checks every few seconds. When IRIS finds a file, it reads it. You can also place IRIS output files in a known directory that is polled by the other program.

When using a polling scheme, first copy the file to the directory with a "." prefix. When the file is fully there, rename it without the ".". This prevents the recipient from seeing a partial file.

User Product Insert (UPI)

User Product Insert (UPI) sends a message to notify the recipient of a product. This removes time lag and CPU usage due to polling.

The user routine communicates with IRIS as in other IRIS host system communication. Because communication is socket-based, the user routine can execute on the same computer as the IRIS host, or it can be on a different CPU.

Vaisala supplies a C++, socket-based example routine.

3.3 Building example programs

The example programs showing the UPI process are shipped with the IRIS release media in the *utils/examples* directory. Compile the programs to learn more.

Table 4 Example programs

Program	Description
upi_rcv	Receives a file from IRIS, prints out the file name, then deletes it. It can be customized to your requirements.
change_product	Modifies an existing product file to add in a circular target.
upi_xmt	Sends files back to IRIS

1. Configure your IRIS development environment.
2. Go to the directory containing the source code: *./iris/examples*
3. Compile the source code from the resident *makefile* with the command: **$ make upi_rcv**
4. To rename the program, copy it elsewhere and make a script file to compile it.

   Take the compile and link commands from the *Makefile* file.

3.4 Running the upi_rcv program

These instructions assume that you are running **upi_rcv** on a computer node called **target**, and that IRIS is running on a computer called **host**.

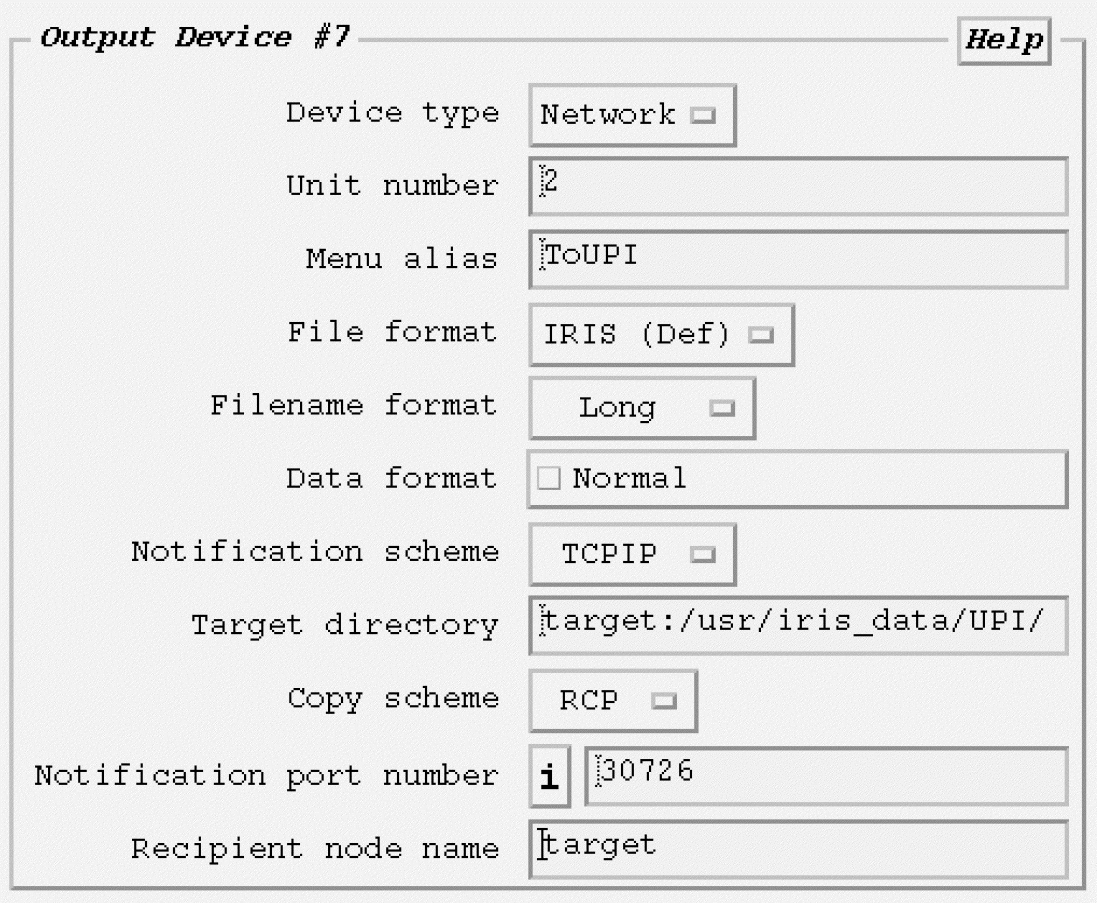
Running on a single computer is simpler.

1. Create the directory on **target** into which IRIS can copy products.

   For example */usr/iris_data/UPI*. The directory name is not important.
2. Copy **upi_rcv** to the target computer (if it is different from where you compiled it) so it can run there.
3. Reset the mode after copying.

4. In the **Setup** utility, on the IRIS host system, create a new network output device called **ToUPI** for sending products to the `upi_rcv` program.

   For example:



5. On the `target` computer, run the upi program:

   `$ upi_rcv -d /usr/iris_data/UPI`
   For example:

```
Initializing UPI receiver
Connection established
Received file: '/usr/iris_data/UPI/host950710171113.PPI'
Connection closed
```

6. In **Setup > Product Output**, send a horizontal reflectivity product to **ToUPI**.

3.5 IRIS product and data types

User-generated products must be one of the IRIS product and data types.

For example, see the display of a typical square product (such as a **PPI**) on a medium sized window:

- IRIS uses the product type for the general the interpretation of the image.
- IRIS uses the data type to determine how to convert between data number and colors.

The definitions for the data type parameters are in the *include/dsp_lib.h* file. See Data types (page 71).

The definitions for the product types are in the *include/product.h* file.

Product Types

The product type in a product file controls some of the text legend as well as the geometry. Based on the product type, IRIS divides products into several geometrical categories which determine what kind of overlays can be drawn on the image:

Table 5 Product Types

Product Type	Description	Example
Horizontal	These products can have political overlays, product overlays, range rings, and latitude/longitude grids drawn on them. There are minor variations within this category, such as constant elevation, constant height, and no height implied.	**CAPPI**
	Constant elevation	**PPI**
	Constant height	**CAPPI**
	No height implied	**Echo tops**
Vertical	These products can have a range/height overlay grid.	**Cross Section**
None	These products have no geometrical interpretation. There are no overlays drawn, and the user cursor does not return information.	**VVP**
No Display	These products cannot be displayed.	**RAW**

Data Types

The data type in a product file controls the color legend (the box containing an array of colored rectangles with text next to them) The data type is converted to text to label the legend, and is used internally to help control how data values in Cartesian products are converted to colors. Some data types do not display a color legend, for example, **WARN**, **SLINE**, **TRACK**, and **VVP** are displayed as graphs.

3.6 Customizing the legend

During operation, when IRIS draws the legend and the product type is `PROD_USER`, `PROD_USERV`, or `PROD_OTHER`, it searches the *out_legend.dat* file. During the search, IRIS identifies the matches:

- If it finds a match of the product type, product name, resolutions, and line number, it is considered a match.
- If it matches everything except the product name, then the first such match in the table is taken as a default for product names, and is considered a match.
- If it would otherwise match, except that the desired resolution is larger than the table, it is considered a match.

The resulting string is displayed on the legend.

You can edit the *out_legend.dat* file to customize the legend on user-defined products by overwriting the following lines

- The text displayed in the 5-line legend box on the window legend.
- The 6 lines above the time on the old-medium sized legend.
- The 4 lines above the time on the old-small sized legend.

1. Edit each line as needed.

   The column headed by **L** stands for line: "1" means line 7 on the text legend, and "2" means line 8 on the text legend.
   The first entry in the table is the product type, which must be **USER**, **VUSER**, or **OTHER**.

2. For each line, in the **R** column, edit the resolution, the number of characters on the line.

   - Low resolution windows can fit 13 characters
   - Medium and high resolution can fit 17 characters

   In general, you must add 2 definitions for each string you want to control. IRIS defaults to the 11-character format for bigger displays if no 17-character entry is defined.

3. For each line, edit the flag values.

   Table 6 out_legend.dat flag values

Flag value	Description
SCALE	A 32-bit word is grabbed from the first byte of the product header **product_specific_info**field offset by the offset in the file. This number is converted to floating point, and multiplied by the scale factor in the file. The C library routine **sprintf** is called with the format statement and the resulting number as arguments.
REF	A pointer to the product header **product_specific_info** field offset by the offset in the file is passed to **sprintf**. This is useful if you want to display a character string from the header.
Neither	If the flag is neither of the above, a 32-bit word is grabbed from the first byte of the product header **product_specific_info** field offset by the offset in the file without any conversion. The C library routine **sprintf** is called with the format statement and the 32-bit number as arguments.

For example:

```
# File: output_legend.dat
# Format:
# Valid flags are: SCALE, REF, <anything else>
# R=resolution, L=line, O=offset
#ptype  productname  R   L  O  scale    flag    format
USER    DEFAULT      11  1  4  0.00001  SCALE   "Hght:%3.1f"
USER    Z_010_120    11  1  4  0.00001  SCALE   "Hght:%3.1f"
USER    DEFAULT      11  2  8  1        -       "Flag:%d"
USER    Z_010_120    11  2  8  1        -       " "
VUSER   DEFAULT      11  1  4  1        -       " "
USER    DEFAULT      17  1  4  0.00001  SCALE   "Height:%3.1f"
USER    Z_010_120    17  1  4  0.00001  SCALE   "Height:%3.1f"
USER    DEFAULT      17  2  8  1        -       "Flag:%d"
USER    Z_010_120    17  2  8  1        -       " "
VUSER   DEFAULT      17  1  4  1        -       " "
```

4. Data formats

4.1 Data format overview

The archived data formats used by IRIS include data storage on disk files, and output tape formats.

The data formats also include antenna, signal processor, and general-purpose constants.

All data format sizes and addresses are given in bytes.

4.2 Scalar definitions

The file `${IRIS_ROOT}/include/sigtypes.h` defines some scalar data `typedefs` that can be used across platforms. The data types are used by the IRIS library routines for defining return values and routine arguments.

Table 7 IRIS data types

Type	Mapped-to	Description
SINT1	int8_t	Signed 8-bit integer
UINT1	uint8_t	Unsigned 8-bit integer
SINT2	int16_t	Signed 16-bit integer
UINT2	uint16_t	Unsigned 16-bit integer
SINT4	int32_t	Signed 32-bit integer
UINT4	uint32_t	Unsigned 32-bit integer
FLT4	float	Floating-point number
FLT8	double	Double-precision floating-point number
BIN2	uint16_t	16-bit binary angle
BIN4	uint32_t	32-bit binary angle
MESSAGE	uint32_t	Encoded error value

A binary angle is an efficient way to store an angle into an integer. In this format the MSB has weight 180°, the next bit means 90°, the next 45°, and so on.

You can convert an N-bit binary angle to degrees with:

```
Degrees = 360 * (Binary Angle) / 2^N
```

To see C language examples of how to do these conversions, see *src/libs/user/angle.c*.

4.3 Structure definitions

The structure definitions are provided in tables with the following information about each structure:

- Source — The name of the include file containing the C structure definition. The source files can be found in the *${IRIS_ROOT}/include* directories.
- Byte — The byte offset of the structure element.
- Size — The data type of the element.
  For structures and spare space, the size in bytes is displayed here. Arrays are designated by a data type followed by a number in square brackets.
- Contents — A brief description of the structure element.
  Structure elements are identified with angle brackets such as <ymds_time>. <spare> indicates reserved space.

4.3.1 beam_psi_struct structure

Source: **headers.h**

Byte	Size	Contents
0	UINT4	Minimum range in cm
4	UINT4	Maximum range in cm
8	BIN4	Left azimuth
12	BIN4	Right azimuth
16	BIN4	Lower elevation
20	BIN4	Upper elevation
24	BIN4	Azimuth smoothing
28	BIN4	Elevation smoothing
32	BIN4	Az of sun at start
36	BIN4	El of sun at start
40	BIN4	Az of sun at end
44	BIN4	El of sun at end

4.3.2 cappi_psi_struct structure

Source: **headers.h**

Byte	Size	Contents
0	4	Shear flags, same as `shear_psi_struct`
4	SINT4	Height of CAPPI (cm above reference)
8	UINT2	Flags: Bit 0= make pseudo CAPPI Bit 1= Velocity is horizontal winds
10	BIN2	Azimuth smoothing for shear
12	char[12]	VVP name to use for shear correction
24	UINT4	Max age for vvp shear correction in seconds

4.3.3 catch_psi_struct structure

Source: `headers.h`

Byte	Size	Contents
0	UINT4	Flags in low 16-bits; From 8.09: RAIN 1 flags in upper 16-bits Bit 0: Enable warning if catchments exceed threshold
4	UINT4	Hours of accumulation
8	SINT4	Threshold offset in 1/1000 or mm
12	SINT4	Threshold faction in 1/1000
16	char[12]	Name of RAIN1 product to use
28	char[16]	Name of catchment file to use
44	UINT4	From 8.09: Seconds of accumulation in low 16-bits
48	UINT4	From 8.09: RAIN1 min Z
52	UINT4	From 8.09: RAIN1 span in seconds
56	UINT4	From 8.09: Average Gage correction factor in low 16-bits

4.3.4 catch_results structure

Source: `product.h`

Byte	Size	Contents
0	char[16]	Name of catchment area, null terminated
16	UINT4	Number of catchment area
20	BIN4	Latitude of label
24	BIN4	Longitude of label

Byte	Size	Contents
28	SINT4	Area of catchment in 1/100 of square km
32	SINT4	Number of pixels in the catchment area
36	SINT4	Number of pixels scanned in the catchment area
40	UINT4	Flags Bit 0: Warning enabled for this catchment Bit 1: Warning triggered for this catchment
44	UINT2	Rainfall accumulation in catchment, DB_FLIQUID2 format
46	UINT2	Rainfall accumulation warning threshold, DB_FLIQUID2
48	52	`<spare>`
100	UINT2[96]	Rainfall accumulation for each input, DB_FLIQUID2 format

4.3.5 color_scale_def structure



The changes in `color_scale_def` are non-critical (unused), for consistency only.

Source: `headers.h`

Byte	Size	Contents

0	UINT4	**iflags:**
		`Bit 8=COLOR_SCALE_VARIABLE`
		`Bit 10=COLOR_SCALE_TOP_SAT`
		`Bit 11=COLOR_SCALE_BOT_SAT`
		`Bit 12=ENUMERATED DATA`

4	SINT4	**istart:** Starting level
8	SINT4	**istep:** Level step
12	SINT2	**icolcnt:** Number of colors in scale
14	UINT2	**iset_and_scale:** Color set number in low byte, color scale number in high byte.
16	UINT2[16]	**ilevel_seams:** Variable level starting values

This structure appears at the end of the `product_configuration` structure. It holds information configured into the product about how quantizing the data into color levels for display.

Vaisala may remove this structure in future because IRIS can override this at display time. This is only used if the user selects **Use Default Scale** in the Color Scale Tool. The bit 11 is used to identify enumerated data type such as `DB_HCLASS (DB_HCLASS2)`, requiring further information about the discrete data values.

To specify the color scale, use one of the following:

- Fixed scale has uniformly spaced numbers fully specified by a start and step value
- Variable scale is specified by a set of 16 individually controlled data seams
- Enumerated data in which each value has an discrete meaning, defined by the string data in enumeration_def structures, which can represent enumeration data as multiples of 14 classes

The **iflags** bits `COLOR_SCALE_VARIABLE`, overruled by `ENUMERATED_DATA` (relates to products using `DB_HCLASS`) indicate which mode is in effect.

In the first two cases, the number of colors can be 2 ... 16, and optionally zoomed or split to give up to 32 colors. True color representation is possible in IRIS. Bits 10 and 11 in **iflags** specify what to do with data off the end of the scale. The choices are:

- saturate at the end (use the last color)
- threshold (do not display)

A typical treatment of reflectivity is to saturate the high end of the scale (so as not to lose the strong features), and threshold at the low end (to inhibit display of very weak speckles).

The number of main colors and labels in the legend is set by **icolcnt**. To select which subset of colors is used for the scale, fill in the low byte of **iset_and_scale**. The upper byte is not used in the product header.

For fixed spacing, set the **istart** and **istep** to the start and step values desired. These values are integers which are 10 times the physical units as discussed in the **color_setup** utility chapter in *IRIS and RDA Utilities Guide (M212925EN)*. The labels of **istart**, `istart+istep`, `istart+2*istep`, and so on appear on the right side of the IRIS legend. Note that:

- Width assumes that the starting value is 0.
- Velocity assumes that the middle of the scale is at 0.
  If the velocity step is set to 0 it means, the spacing automatically scales to fit the Nyquist velocity.

For variable spacing (bit 8 ON), the numbers in the array **ilevel_seams** control the starting values for each of the colors. The first seam is the bottom of the first color. The top of the 16th color is computed by adding the step between the 15th and 16th seam to the 16th seam. These seams are data values, using the 16-bit versions of the data. For shear and height data, for which there is no 16-bit version, the 8-bit version is used.

Enumerated data (bit 12 ON) can be treated as other data types, that is either with fixed spacing or variable spacing.

4.3.6 cross_psi_struct structure

Source: `headers.h`

Byte	Size	Contents
0	BIN2	Azimuth angle of line from left to right
2	10	<spare>
12	SINT4	East coordinate of center (in cm relative to radar)
16	SINT4	North coordinate of center (in cm relative to radar)
20	SINT4	User miscellaneous

4.3.7 dsp_data_mask structure

Source: `sigtypes.h`

Byte	Size	Contents	Description
0	UINT4	Mask word 0	
4	UINT4	Extended header type	
8	UINT4	Mask word 1	Contains bits set for all recorded data.
12	UINT4	Mask word 2	See the `DB_*` parameter in Table 18 (page 100) for bit specification.
16	UINT4	Mask word 3	
20	UINT4	Mask word 4	

4.3.8 extended_header_v0 structure

Source: `ingest.h`

Byte	Size	Contents
0	SINT4	Time in milliseconds from the sweep starting time
4	SINT2	Calibration Signal level
6	14	<spare>

4.3.9 extended_header_v1 structure

Source: `ingest.h`

Byte	Size	Contents
0	SINT4	Time in milliseconds from the sweep starting time
4	SINT2	Calibration Signal level
6	BIN2	Azimuth (binary angle)
8	BIN2	Elevation (binary angle)

Byte	Size	Contents
10	BIN2	Train Order (binary angle)
12	BIN2	Elevation Order (binary angle)
14	BIN2	Pitch (binary angle)
16	BIN2	Roll (binary angle)
18	BIN2	Heading (binary angle)
20	BIN2	Azimuth Rate (binary angle/second)
22	BIN2	Elevation Rate (binary angle/second)
24	BIN2	Pitch Rate (binary angle/second)
26	BIN2	Roll Rate (binary angle/second)
28	BIN4	Latitude (binary angle)
32	BIN4	Longitude (binary angle)
36	BIN2	Heading Rate (binary angle/second)
38	SINT2	Altitude (meters above MSL)
40	SINT2	Velocity East (cm/second)
42	SINT2	Velocity North (cm/second)
44	SINT4	Time since last update (milliseconds)
48	SINT2	Velocity Up (cm/second)
50	UINT2	Navigation System OK flag
52	SINT2	Radial velocity correction (same units as velocities)

4.3.10 extended_header_v2 structure

Source: none

Byte	Size	Contents
0	SINT4	Time in milliseconds from the sweep starting time
4	SINT2	Calibration Signal level
6	2	`<spare>`
8	SINT4	Number of bytes in the header
12	?	Remainder customer specified

4.3.11 fcast_psi_struct structure

Source: `headers.h`

Byte	Size	Contents
0	UINT4	Correlation threshold, 0 ...100
4	SINT4	Data threshold
8	SINT4	Mean speed (cm/hour), 0 if none
12	BIN4	Direction of mean speed
16	UINT4	Maximum time between inputs (seconds)
20	SINT4	Maximum allowable velocity (mm/sec)
24	UINT4	Flags
28	SINT4	Desired output resolution (cm)
32	UINT4	Type of input product
36	char[12]	Name of input product (space padded)

4.3.12 gage_psi_struct structure

Source: `headers.h`

Byte	Size	Contents
0	UINT4	Time span of the rain gage data in seconds.
4	UINT4	Flag bits: 0 = File has distrometers 1 = File does not have gages

4.3.13 gage_results structure

Source: `product.h`

Byte	Size	Contents
0	char[16]	Name of raingage, null terminated
16	BIN32_T	Latitude of raingage
20	BIN32_T	Longitude of raingage
24	uint16_t	Average rainrate in DB_RAINRATE2 format for the time span
26	uint16_t	Correction factor last calculated in DB_CDBZ2 format
28	uint8_t	Data Quality, range 0-10
29	uint8_t	Flag bits: 0 = Skip this gage for correction calculation 1 = Gage has Z/R numbers 2 = Gage does not have rainrate 3 = Radar rainrate filled in
30	uint16_t	Z/R constant in 1/10

Byte	Size	Contents
32	uint16_t	Z/R exponent in 1/1000
34	uint16_t	Average radar rainrate in DB_RAINRATE2 format for the span
36	4	<spare>

4.3.14 ingest_configuration structure

Source: `ingest.h`

Byte	Size	Contents
0	char[80]	Name of file on disk
80	SINT2	Number of associated data files extant
82	SINT2	Number of sweeps completed so far
84	SINT4	Total size of all files in bytes
88	12	`<ymds_time>` Time that volume scan was started, TZ spec in bytes 166 & 224
100	112	`<spare>`
112	SINT2	Number of bytes in the ray headers
114	SINT2	Number of bytes in extended ray headers (includes normal ray header)
116	SINT2	Number of task configuration table
118	SINT2	Playback version number
120	4	`<spare>`
124	char[8]	IRIS version, null terminated
132	char[16]	Hardware name of site
148	SINT2	Time zone of local standard time, minutes west of GMT
150	char[16]	Name of site, from setup utility
166	SINT2	Time zone of recorded standard time, minutes west of GMT
168	BIN4	Latitude of radar (binary angle: 20000000 hex is 45º North)
172	BIN4	Longitude of radar (binary angle: 20000000 hex is 45º East)
176	SINT2	Height of ground at site (meters above sea level)
178	SINT2	Height of radar above ground (meters)
180	UINT2	Resolution specified in number of rays in a 360º sweep
182	UINT2	Index of first ray from above set of rays
	184UINT2	Number of rays in a sweep
186	SINT2	Number of bytes in each gparam

Byte	Size	Contents
188	SINT4	Altitude of radar (cm above sea level)
192	SINT4[3]	Velocity of radar platform (cm/sec) (east, north, up)
204	SINT4[3]	Antenna offset from INU (cm) (starboard, bow, up)
216	UINT4	Fault status at the time the task was started, bits: 0:Normal BITE 1:Critical BITE 2:Normal RCP 3:Critical RCP 4:Critical system 5:Product gen. 6:Output 7:Normal system
220	SINT2	Height of melting layer (meters above sea level) MSB is complemented, 0=Unknown
222	2	`<spare>`
224	char[8]	Local timezone string, null terminated
232	UINT4	Flags, Bit 0=First ray not centered on 0°
226	char[16]	Configuration name in the *`dpolapp.conf`* file, null terminated
252	228	`<spare>`

4.3.15 ingest_data_header structure

Source: **`ingest.h`**

Byte	Size	Contents

0	12	`<structure_header>`
		The size stored here is the total size of the file:
		`76 + 4*Irtotl + Iwritn*raysize`

12	12	`<ymds_time>` Date and time the sweep started TZ specified in *`ingest_configuration`* See ymds_time structure (page 71).
24	SINT2	Sweep number, origin 1
26	SINT2	Resolution specified in number of rays in a 360º sweep
28	SINT2	Index of first ray from above set of rays
30	SINT2	**`Irtotl`** Number of rays (and pointers) expected in the file

Byte	Size	Contents
32	SINT2	**Iwritn** Number of rays actually in the file
34	BIN2	Fixed angle for this sweep (binary angle)
36	SINT2	Number of bits per bin for this data type
38	UINT2	Data type in the file: 0=extended header 1=Total power (dBZ) 2=Reflectivity (dBZ) 3=Velocity 4=Width 5=ZDR
40	36	<spare>

4.3.16 ingest_header structure

A file containing the **ingest_header** structure is written for each volume scan.

This file is updated each time a sweep finishes. This allows product generators to get data from the ingest files as soon as a sweep is completed.

Source: **ingest.h**

<structure_header> 12 Bytes
<ingest_configuration> 480 Bytes
<task_configuration> 2612 Bytes
Spare 732 Bytes
GPARM from DSP 128 Bytes
Reserved 920 Bytes

4.3.17 max_psi_struct structure

Source: **headers.h**

Byte	Size	Contents
0	4	<spare>
4	SINT4	Bottom of interval in cm
8	SINT4	Top of interval in cm
12	SINT4	Number of pixels in side panels
16	SINT2	Horizontal smoother in side panels
18	SINT2	Vertical smoother in side panels

4.3.18 mlhgt_psi_struct structure

Source: `headers.h`

Byte	Size	Contents
0	unit32_t	Flag word: Bit 0: localize observed BB altitudes Bit 1: the product is processed from RHI scans Bit 2: extrapolate promptly to fair weather areas Bit 3: arrangement to display confidence levels interleaved with data
4	int16_t	Momentary ML altitude averaged through the product area [m] (MSL)
8	int16_t	Interval of acceptable ML altitudes, from climatology [m]
12	int16_t	Vertical spacing in the likelihood grid [m]
16	uint16_t	Number of Az sectors in the internal polar grid
20	uint32_t	Relaxation time of the MLHGT confidence (exponential decay) [seconds]
24	uint16_t	Modeled fraction of the correct Melt classifications [1/(256*256)]
28	uint16_t	Modeled fraction of the correct NoMelt classifications [1/(256*256)]
32	uint16_t	Minimum confidence for acceptance to a local cluster [DB_LDR]

4.3.19 ndop_input structure

Source: `headers.h`

Byte	Size	Contents
0	char[12]	Task name
12	char[3]	Site code
15	UINT1	Flags

4.3.20 ndop_psi_struct structure

Source: `headers.h`

Byte	Size	Contents
0	3*16	`<ndop_input>`
48	SINT4	Time window
52	SINT4	CAPPI height (cm above reference)
56	SINT4	Output resolution (cm)
60	BIN4	Minimum permitted crossing angle

Byte	Size	Contents
64	UINT4	Flags: Bit 0=Make diagnostic products
68	char[4]	Output site code, added in 8.09.6, use first input if zeroed
72	8	<spare>

4.3.21 ndop_results structure

Used for both the NDOP and FCAST products. The change rate is always zero in NDOP products. The Signal Quality Index (SQI) is unused in FCAST products. The SQI is a function of the variance of the calculated velocity normalized such that random vectors at half the Nyquist velocity would produce an SQI of zero.

$$SQI = 1 - \left[\sqrt{\frac{VarianceEast + VarianceNorth}{2}}\right] / Vnorm$$

Source: `product.h`

Byte	Size	Contents
0	UINT2	Velocity East in cm/second (DB_VEL2 format)
2	UINT2	Velocity North in cm/second
4	UINT2	Change rate in db/min (DB_CDBZ2 format)
6	UINT1	Signal Quality Index * 256
7	5	<spare>

4.3.22 one_protected_region structure

Source: `setup.h`

Byte	Size	Contents
0	SINT4	East center from radar in cm
4	SINT4	North center from radar in cm
8	SINT4	East-West size in cm
12	SINT4	North-South size in cm
16	UINT2	Orientation angle in binary angle
18	2	<spare>
20	char[12]	Name of the region, all spaces means unused

4.3.23 ppi_psi_struct structure

Source: `headers.h`

Byte	Size	Contents
0	BIN2	Elevation angle

4.3.24 product_configuration structure

Source: `product.h`

Byte	Size	Contents
0	12	`<Structure_Header>`

Byte	Size	Contents
12	UINT2	Product type code: 1:PPI 2:RHI 3:CAPPI 4:CROSS 5:TOPS 6:TRACK 7:RAIN1 8:RAINN 9:VVP 10:VIL 11:SHEAR 12:WARN 13:CATCH 14:RTI 15:RAW 16:MAX 17:USER 18:USERV 19:OTHER 20:STATUS 21:SLINE 22:WIND 23:BEAM 24:TEXT 25:FCAST 26:NDOP 27:IMAGE 28:COMP 29:TDWR 30:GAGE 31:DWELL 32:SRI 33:BASE 34:HMAX 35:VAD 36:THICK 37:SATELLITE 38:LAYER 39:SWS 40:MLGHT

Byte	Size	Contents
14	UINT2	Scheduling code: 0:hold 1:next 2:all
16	SINT4	Number of seconds to skip between runs
20	12	<ymds_time> Time product was generated (UTC). See ymds_time structure (page 71).
32	12	<ymds_time> Time of input ingest sweep (TZ flex). See ymds_time structure (page 71).
44	12	<ymds_time> Time of input ingest file (TZ flexible) See ymds_time structure (page 71).
56	6	<spare>
62	char[12]	Name of the product configuration file
74	char[12]	Name of the task used to generate the data
86	UINT2	Flag word: (Bits 0,2,3,4,8,9,10 used internally) Bit1: TDWR style messages Bit5: Keep this file Bit6: This is a clutter map Bit7: Speak warning messages Bit11: This product has been composited Bit12: This product has been dwelled Bit13: Z/R source0, 0:Type-in; 1:Setup; 2:Disdrometer Bit14: Z/R source1
88	SINT4	X scale in cm/pixel
92	SINT4	Y scale in cm/pixel
96	SINT4	Z scale in cm/pixel
100	SINT4	X direction size of data array
104	SINT4	Y direction size of data array
108	SINT4	Z direction size of data array
112	SINT4	X location of radar in data array (signed 1/1000 of pixels)
116	SINT4	Y location of radar in data array (signed 1/1000 of pixels)
120	SINT4	Z location of radar in data array (signed 1/1000 of pixels)
124	SINT4	Maximum range in cm (used only in version 2.0, raw products)
128	2	<spare>

Byte	Size	Contents
130	UINT2	Data type generated. See Constants (page 100).
132	char[12]	Name of projection used
144	UINT2	Data type used as input. See Constants (page 100).
146	UINT1	Projection type: 0=Centered Azimuthal 1=Mercator
147	1	<spare>
148	SINT2	Radial smoother in 1/100 of km
150	SINT2	Number of times this product configuration has run
152	SINT4	Z/R relationship constant in 1/1000
156	SINT4	Z/R relationship exponent in 1/1000
160	SINT2	X-direction smoother in 1/100 of km
162	SINT2	Y-direction smoother in 1/100 of km
164	80	<product_specific_info>
244	char[16]	List of minor task suffixes, null terminated
260	12	<spare>
272	48	<color_scale_def> Color scale definition. May be removed in the future. See color_scale_def structure (page 27).

4.3.25 product_end structure

Source: **product.h**

Byte	Size	Contents
0	char[16]	Site name — where product was made (space padded)
16	char[8]	IRIS version where product was made (null terminated)
24	char[8]	IRIS version where ingest data came from
32	12	<ymds_time> Time of oldest input ingest file (only **RAIN1** and **RAINN**, **TZ** flexible)
44	28	<spare>
72	SINT2	Number of minutes local standard time is west of GMT
74	char[16]	Hardware name where ingest data came from (space padded)
90	char[16]	Site name where ingest data came from (space padded)
106	SINT2	Number of minutes recorded standard time is west of GMT

Byte	Size	Contents
108	BIN4	Latitude of center (binary angle)[1)]
112	BIN4	Longitude of center (binary angle)[1)]
116	SINT2	Signed ground height in meters relative to sea level
118	SINT2	Height of radar above the ground in meters
120	SINT4	PRF in hertz
124	SINT4	Pulse width in 1/100 of microseconds
128	UINT2	Type of signal processor used
130	UINT2	Trigger rate scheme
132	SINT2	Number of samples used
134	char[12]	Clutter filter file name
146	UINT2	Number of linear based filter for the first bin
148	SINT4	Wavelength in 1/100 of centimeters
152	SINT4	Truncation height (cm above the radar)
156	SINT4	Range of the first bin in cm
160	SINT4	Range of the last bin in cm
164	SINT4	Number of output bins
168	UINT2	Flag wordBit0: Disdrometer failed. Used setup for Z/R source instead
170	SINT2	Number of ingest or product files used to make this product (only on **RAIN1** and **RAINN**)
172	UINT2	Type of polarization used
174	SINT2	I0 cal value, horizontal pol, in 1/100 dBm
176	SINT2	Noise at calibration, horizontal pol, in 1/100 dBm
178	SINT2	Radar constant, horizontal pol, in 1/100 dB
180	UINT2	Receiver bandwidth in kHz
182	SINT2	Current noise level, horizontal pol, in 1/100 dBm
184	SINT2	Current noise level, vertical pol, in 1/100 dBm
186	SINT2	LDR offset, in 1/100 dB
188	SINT2	ZDR offset, in 1/100 dB
190	uint16_t	TCF Cal flags, see struct task_calib_info (added in 8.12.3)
192	uint16_t	TCF Cal flags2, see struct task_calib_info (added in 8.12.3)
194	uint8_t	if enumerated data (DB_HCLASS) the identifier of classifier in the first bit segment else **<spare>**
195	uint8_t	if enumerated data (DB_HCLASS) the identifier of classifier in the 2nd bit segment else **<spare>**

Byte	Size	Contents
196	uint8_t	if enumerated data (DB_HCLASS) the identifier of classifier in the 3rd bit segment else <spare>
197	uint8_t	if enumerated data (DB_HCLASS2) the identifier of the 4th classifier else <spare>
198	uint8_t	if enumerated data (DB_HCLASS2) the identifier of the 5th classifier else <spare>
199	uint8_t	if enumerated data (DB_HCLASS2) the identifier of the 6th classifier else <spare>
200	12	<spare>
212	BIN4	More projection info these 4 words: Standard parallel #1
216	BIN4	Standard parallel #2
220	UINT4	Equatorial radius of the earth, cm (0 = 6371km sphere)
224	UINT4	1/Flattening in 1/1000000 (0 = sphere)
228	UINT4	Fault status of task, see ingest_configuration structure (page 32).
232	UINT4	Mask of input sites used in a composite
236	UINT2	Number of log based filter for the first bin
238	UINT2	Nonzero if cluttermap applied to the ingest data
240	BIN4	Latitude of projection reference[1)]
244	BIN4	Longitude of projection reference[1)]
248	SINT2	Product sequence number
250	32	<spare> (used before 8.11.4)
282	SINT2	Melting level in meters, msb complemented (0=unknown)
284	SINT2	Height of radar above reference height in meters
286	SINT2	Number of elements in product results array
288	UINT1	Mean wind speed
289	BIN1	Mean wind direction (unknown if speed and direction 0)
290	2	<spare>
292	char[8]	TZ Name of recorded data
300	UINT4	Offset to extended product header from file (added in 8.13.0)
304	4	<spare>

1) *Note on Latitude and longitudes: Interpretation varies with product type. They are as documented for CAPPI, FCAST, MAX, NDOP, PPI, RAIN1, RAINN, SHEAR, SLINE, TOPS, TRACK, VIL, USER and WARN. For all other products, the Center location is the radar location, and the reference location is 0.*

4.3.26 product_hdr structure

Source: `product.h`

Structure	More information
`<structure_header>` 12 Bytes	structure_header structure (page 55)
`<product_configuration>` 320 Bytes See .	product_configuration structure (page 37)
`<product_end>` 308 Bytes.	product_end structure (page 40)
Data Size Varies	

IRIS allows data to be recorded in UTC on computers with a timezone set to local time. IRIS also records timezone information about the local computer to support optional displaying of different times at output time.

Table 8 IRIS Timezone Recording

System	Record UTC in setup	Record Local in setup

`ComputertimezoneUTC`	`iMinutesWest=0`
	`iLocalWest=0s`
	`LocalTZName="UTC"`
	`UTC Flag=1`
	`DST Flag=0`
	`LDST Flag=0`
	`Default Display is "UTC"`
		`iMinutesWest=0`
		`iLocalWest=0s`
		`LocalTZName="UTC"`
		`UTC Flag=0`
		`DST Flag=0L`
		`DST Flag=0`
		`Default Display is "UTC"`

`ComputertimezoneEST`	`iMinutesWest=0`
	`iLocalWest=300s`
	`LocalTZName="EST"`
	`UTC Flag=1`
	`DST Flag=0`
	`LDST Flag=0`
	`Default Display="UTC"`
		`iMinutesWest=300`
		`iLocalWest=300s`
		`LocalTZName="EST"`
		`UTC Flag=0`
		`DST Flag=0LDST`
		`Flag=0`
		`Default Display is "EST"`

`ComputertimezoneEDT`	`iMinutesWest=0`
	`iLocalWest=300s`
	`LocalTZName="EDT"`
	`UTC Flag=1`
	`DST Flag=0`
	`LDST Flag=1`
	`Default Display is "UTC"`
		`iMinutesWest=300`
		`iLocalWest=300`
		`LocalTZName="EDT"`
		`UTC Flag=0`
		`DST Flag=1`
		`!LDST Flag=1Default Display is "EDT"`
		Do not record summer time.

4.3.27 product_header_extensions

If the `hdr.end.iExtendedHeaderOffset` is non-zero, then there is a product header extension. This number is the byte offset from the beginning of the file to the extension.

The product header extension is an ASCII format, containing meta-data in name=value pairs.

The name value pairs are separated by newline characters.

The ASCII meta-data section is terminated by an ASCII zero.

This null may be followed by more null, and ends with a `0xff`. This allows a parser to find the end by looking for a final `0x00` followed by a `0xff`, and allows the sized to be padded to make the next section aligned, if desired.

Following the product header extension, there may be a binary data section. This binary data section is aligned based on the size of the data elements, which is specified in the meta-data. This is a general design intended to handle many needs for extending headers, such as listing sites in a composite, or listing algorithm attributes used to generate a data moment. In some cases, there may be multiple product header extension sections.

For example, the echo thickness product stores 2 binary data arrays, first the thickness, and second the average reflectivity over that interval. See, for example, the following name value pairs:

```
“data_type=Z”, “data_units=dBZ”, “x_size=120”, “y_size=120”,
“data_storage=uint8_t”, “data_bits=8”, “data_source=DB_DBZ”, “data_size=14400”
```

A binary blob containing the average reflectivity follows the extension .

4.3.28 product_specific_info structure

Source: `headers.h`

Byte	Size	Contents	Product
0	24	`<beam_psi_struct beam>`	BEAM
0	10	`<cappi_psi_struct>`	CAPPI
0	44	`<catch_psi_struct>`	CATCH
0	80	`<comp_psi_struct>`	COMP
0	80	`<cross_psi_struct>`	USERV, XSECT
0	68	`<dwell_psi_struct>`	DWELL
0	48	`<fcast_psi_struct>`	FCAST
0	8	`<gage_psi_struct>`	GAGE
0	20	`<maximum_psi_struct>`	MAX
0	28	`<mlhgt_psi_struct>`	MLGHT
0	72	`<ndop_psi_struct>`	NDOP

Byte	Size	Contents	Product
0	2	<ppi_psi_struct>	PPI
0	24	<rain_psi_struct>	RAIN1, RAINN
0	20	<raw_psi_struct>	RAW
0	2	<rhi_psi_struct>	RHI
0	20	<rti_psi_struct>	RTI
0	80	<sat_psi_struct>	SATELLITE
0	28	<shear_psi_struct>	SHEAR
0	60	<sline_psi_struct>	SLINE
0	34	<sri_psi_struct>	SRI
0	8	<sws_psi_struct>	SWS
0	14	<tdwr_psi_struct>	TDWR
0	6	<top_psi_struct>	BASE, HMAX, THICK, TOPS
0	52	<track_psi_struct>	TRACK
0	80	<user_psi_struct>	USER
0	16	<vad_psi_struct>	VAD
0	4	<vil_psi_struct>	VIL
0	28	<vvp_psi_struct>	VVP
0	80	<warn_psi_struct>	WARN
0	40	<wind_psi_struct>	WIND
0	80	<spare>	OTHER, TEXT

4.3.29 protect_setup structure

Source: `setup.h`

Byte	Size	Contents
0	1024	<one_protected_region> Protected region definitions

4.3.30 rain_psi_struct structure

Source: `headers.h`

Byte	Size	Contents
0	UINT4	Minimum Z to accumulate (in 2-byte Reflectivity Format)
4	UINT2	Average gage correction factor

Byte	Size	Contents
6	UINT2	Seconds of accumulation
8	UINT2	Flag word: Bit0: Apply clutter map Bit1: Apply gage correction Bit2: Clutter map was applied Bit3: Gage correction was applied
10	SINT2	Number of hours to accumulate (RAINN only)
12	char[12]	Name of input product to use (space padded)
24	UINT4	Span in seconds of the input files

4.3.31 raw_prod_bhdr structure

Source: `product.h`

Byte	Size	Contents
0	SINT2	Record number within the file (origin 0: record 3 contains a 2)
2	SINT2	Sweep number (1 is first sweep)
4	SINT2	Byte offset of first full ray in this record (-1 if none)
6	SINT2	Ray number within sweep for above pointed to ray
8	UINT2	Flags
10	2	`<spare>`

4.3.32 raw_psi_struct structure

Source: `headers.h`

Byte	Size	Contents
0	UINT4	Data type mask word 0
4	SINT4	Range of last bin in cm
8	UINT4	Format conversion flag: 0=Preserve all ingest data 1=Convert 8-bit data to 16-bit data 2=Convert 16-bit data to 8-bit data
12	UINT4	Flag word: Bit 0=Separate product files by sweep Bit 1=Mask data by supplied mask
16	SINT4	Sweep number if separate files, origin 1

Byte	Size	Contents
20	4	Xhdr type (unused)
24	4	Data type mask 1
28	4	Data type mask 2
32	4	Data type mask 3
36	4	Data type mask 4
40	4	Playback version (low 16-bits)

4.3.33 ray_header structure

Source: `ingest.h`

Byte	Size	Contents
0	BIN2	Azimuth at beginning of ray (binary angle)If dual-PRF: bit 0=ray's PRF was high
2	BIN2	Elevation at beginning of ray (binary angle)If trigger blanking on: bit 0=Trigger was not blanked
4	BIN2	Azimuth at end of ray (binary angle)
6	BIN2	Elevation at end of ray (binary angle)
8	SINT2	Actual number of bins in the ray
10	UINT2	Time in seconds from start of sweep (unsigned)

4.3.34 rhi_psi_struct structure

Source: `headers.h`

Byte	Size	Contents
0	BIN2	Azimuth angle

4.3.35 rti_psi_struct structure

Source: `headers.h`

Byte	Size	Contents
0	BIN4	Nominal sweep angle
4	UINT4	Starting time offset from sweep time, ms
8	UINT4	Ending time offset
12	BIN4	Azimuth of the first ray in the file

Byte	Size	Contents
16	BIN4	Elevation of the first ray in the file

4.3.36 shear_psi_struct structure

Source: `headers.h`

Byte	Size	Contents
0	BIN4	Azimuthal smoothing angle
4	BIN2	Elevation angle
6	2	`<spare>`
8	UINT4	Flag word: Bit0=Do radial shear Bit1=Do azimuthal shear Bit2=Do mean wind correction to azimuthal shear using VVP Bit3=Mean wind correction to azimuthal shear done Bit4=Unfolding done in associated VVP product Bit5=Do elevation shear
12	char[12]	Name of VVP product to use (space padded)
24	UINT4	Maximum age of VVP to use (seconds)

4.3.37 sline_psi_struct structure

Source: `headers.h`

Byte	Size	Contents
0	SINT4	Area in square meters
4	SINT4	Shear threshold (cm/sec/km)
8	UINT4	Bit flags to choose protected areas
12	SINT4	Maximum forecast time in seconds
16	UINT4	Maximum age between products for motion calculation
20	SINT4	Maximum velocity allowed (mm/sec)

Byte	Size	Contents
24	UINT4	Flag word: Bit0=Do radial shear Bit1=Do azimuthal shear (both bits mean do combined) Bit2=Do mean wind correction to azimuth shear using VVP Bit3=Mean wind correction to azimuthal shear done Bit4=Unfolding done in associated VVP product Bit8=Use two elevation angles Bit9=Generate diagnostic output Bit10=Max centroid count exceeded
28	BIN4	Azimuthal smoothing angle (0=none)
32	BIN4	Elevation binary angle
36	BIN4	Elevation binary angle
40	char[12]	Name of VVP task
52	UINT4	Maximum age of VVP in seconds
56	SINT4	Curve fit standard deviation threshold in cm.
60	UINT4	Low byte: Min length of sline (unsigned 1/10 km)

4.3.38 sline_results structure

Source: `product.h`

Byte	Size	Contents
0	SINT4	East coordinate of center point (cm)
4	SINT4	North coordinate of center point (cm)
8	BIN4	Rotation angle to X-Y coordinates of curve fit polynomial Span is -90 … +90°.
12	SINT4	X coordinate of left of curve (cm)
16	SINT4	X coordinate of right of curve (cm)
20	SINT4[6]	6 polynomial coefficients
44	SINT4	Standard deviation of fit
48	SINT4	Propagation speed (mm/second)
52	BIN4	Propagation direction (binary angle)
56	SINT4	Reference side wind speed (mm/second)
60	BIN4	Reference side wind direction (binary angle)
64	SINT4	Other side wind speed (mm/second)
68	BIN4	Other side wind direction (binary angle)

Byte	Size	Contents
72	SINT4[32]	ETA in seconds for each protected area (0 if in area, -1 if unexpected)
200	UINT4	Flags: Bit0=Propagation speed available Bit1=Wind speeds valid
204	796	<spare>

The polynomial curve fit is calculated as follows:

1. Threshold a shear product based on the shear threshold specified in the product configuration.
   Let `East` and `North` refer to the distance of the center of each pixel from the radar position in centimeters. This coordinate system is rotated by an angle θ clockwise about the radar location to produce a new coordinate system with distances also in centimeters. This is the *Rotation angle to X-Y coordinates of curve fit polynomial*. This coordinate system uses the variables `X` and `Y`. The equations for the transformation are:

   ```
   X = Northsinθ + East cosθ
   Y = Northcosθ – East sinθ
   ```

   This transformation is T_{rotate}, and the reverse transformation is T^{-1}_{rotate}.
   In this coordinate system, the `X`-coordinate of the left most end of the line is X_l and the right most end X_r.

2. Shift and scale the coordinate system to keep the polynomial coefficients from becoming too large. The equations for the transformation are:

$$X' = \frac{X - X_l}{X_r - X_l}$$

Call this transformation T_{scale}, and the reverse transformation T^{-1}_{scale}.
In this new coordinate system, the polynomial is expressed as:

$$Y' = A_0 + A_1X' + A_2X'^2 + A_3X'^3 + A_4X'^4 + A_5X'^5$$

or

$$Y' = P[X']$$

The standard deviation is computed as follows: Let $[X'_i, Y'_i]$ represent the i^{th} point in the data set in the rotated and scaled coordinate system, and N the total number of points, then the standard deviation is:

$$standard\ deviation = \sqrt{\frac{1}{N} \sum_{i\ =\ 1,\ N} [Y'i - P[X'i]]^2}$$

The center point is computed as follows: Let $[East_i, North_i]$ represent the i^{th} point in the data set in the original coordinate system, and N the total number of points:

$$East\ center = \sqrt{\frac{1}{N} \sum_{i\ =\ 1,\ N} East_i}$$

$$North\ center = \sqrt{\frac{1}{N} \sum_{i\ =\ 1,\ N} North_i}$$

4.3.39 sri_psi_struct structure

Source: `headers.h`

Byte	Size	Contents
0	UINT4	Flags Bit 0=Do profile correction Bit 3 and 4: Melt src: • 0=Ingest • 1=Setup • 2=TypeIn Bit 5=Check for convection
4	SINT4	Total number of bins inserted
8	SINT4	Number of bins with data
12	SINT4	Number of data bins profile corrected
16	SINT2	Surface height (m above reference)
18	SINT2	Maximum height (m above reference)
20	SINT2	Melting height (m above MSL)
22	SINT2	Melting level thickness (m)
24	SINT2	Gradient above melting (1/100 dB/km)
26	SINT2	Gradient below melting (1/100 dB/km)
28	SINT2	Convective check height (m above melting)
30	SINT2	Convective check level (`DB_DBZ2` format)

4.3.40 status_antenna_info structure

Source: `product.h`

Byte	Size	Contents
0	BIN4	Azimuth position
4	BIN4	Elevation position
8	UINT4	Azimuth velocity
12	UINT4	Elevation velocity
16	UINT4	Command bits
20	UINT4	Command bit availability mask
24	UINT4	Status bits
28	UINT4	Status bit availability mask
32	UINT4	Bite fault flag, 0=OK, 1=Fault, 2=Critical
36	SINT4	Lowest field number generating a fault
40	UINT4	Status bits which can cause critical faults

Byte	Size	Contents
UINT4[3]	Mask indicating the state of each BITE field	44
56	UINT4[3]	Mask indicating which BITE fields are faulted
68	32	<spare>

4.3.41 status_device_info structure

Source: product.h

Byte	Size	Contents
0	40	<status_one_device> dsp
40	40	<status_one_device> antenna
80	720	<status_one_device>[18] output devices

4.3.42 status_message_info structure

Source: product.h

Byte	Size	Contents
0	SINT4	Message count
4	MESSAGE	Actual message number
8	SINT4	Number of times it was repeated
12	char[16]	Process name, null terminated
28	char[80]	Text of message, null terminated
108	char[32]	Name of signal, null terminated
140	20	<serv_ymds_time> Time of message (TZ flexible)
160	UINT4	Message type: 1=info, 2=normal, 3=say
164	36	<spare>

4.3.43 status_misc_info structure

Source: product.h

Byte	Size	Contents
0	char[16]	Radar status configuration name, null terminated
16	char[16]	Task configuration name, null terminated

Byte	Size	Contents
32	char[16]	Product scheduler configuration name, null terminated
48	char[16]	Product output configuration name, null terminated
64	char[16]	Active task name, null terminated
80	char[16]	Active product name, null terminated
96	UINT4	Site type (IRIS style)
100	SINT4	Number of incoming network connects
104	SINT4	Number of IRIS clients connected
108	4	`<spare>`
112	SINT4	Number of output devices
116	UINT4	Flags: bit 0=Automatic mode switching is enabled bit 1=Regular Fault bit 2=Critical Fault bit 3=Regular message fault bit 4=Critical message fault
120	char[4]	Node status fault site
124	12	`<ymds_time>` Time of active task (TZ flexible)
136	64	`<spare>`

4.3.44 status_one_device structure

Source: **`product.h`**

Byte	Size	Contents
0	UINT4	Device type
4	SINT4	Unit number
8	UINT4	Status
12	4	`<spare>`
16	UINT4	Mode from process table
20	char[16]	String (null terminated)
36	4	<spare>

4.3.45 status_one_process structure

Source: **`product.h`**

Byte	Size	Contents
0	UINT4	Command
4	UINT4	Mode
8	12	<spare>

4.3.46 status_process_info structure

Source: `product.h`

Byte	Size	Contents
0	20	<status_one_process> Ingest process
20	20	<status_one_process> Ingfio process
40	20	<spare>
60	20	<status_one_process> Output master process
80	20	<status_one_process> Product process
100	20	<status_one_process> Watchdog process
120	20	<status_one_process> Reingest process
140	20	<status_one_process> Network process
160	20	<status_one_process> Nordrad process
180	20	<status_one_process> Server process
200	20	<status_one_process> RibBuild process
220	180	<spare>

4.3.47 status_results structure

Source: `product.h`

Byte	Size	Contents
0	200	<status_misc_info> Miscellaneous info
200	400	<status_process_info> Status of processes
600	800	<status_device_info> Status of devices
1400	100	<status_antenna_info> Status of antenna
1500	200	<status_message_info> Message information

4.3.48 structure_header structure

Source: `headers.h`

Byte	Size	Contents
0	SINT2	Structure identifier:Value; `Description` 22 `Task_configuration` 23 `Ingest_header` 24 `Ingest_data_header` 25 `Tape_inventory` 26 `Product_configuration` 27 `Product_hdr` 28 `Tape_header_record`
2	SINT2	Format version number (see *`headers.h`*)
4	SINT4	Number of bytes in the entire structure
8	SINT2	Reserved
10	SINT2	Flags: bit 0=structure complete

4.3.49 tape_header_record structure

Source: `output.h`

Byte	Size	Contents
0	12	`<structure_header>`
12	char[16]	Tape identification name
28	char[16]	Name of site that created the tape
44	12	`<ymds_time>` Time that the tape was created (TZ flexible)
56	4	`<spare>`
60	char[8]	IRIS version when the tape was initialized
68	252	`<spare>`

4.3.50 task_calib_info structure

Source: `iris_task.h`

Byte	Size	Contents
0	SINT2	Reflectivity slope (4096*dB/ A/D count)
2	SINT2	Reflectivity noise threshold (1/16 dB above Noise)
4	SINT2	Clutter Correction threshold (1/16 dB)
6	SINT2	SQI threshold (0–1)*256
8	SINT2	Power threshold (1/16 dBZ)

Byte	Size	Contents
10	8	<spare>
18	SINT2	Calibration Reflectivity (1/16 dBZ at 1 km)
20	UINT2	Threshold flags for uncorrected reflectivity
22	UINT2	Threshold flags for corrected reflectivity
24	UINT2	Threshold flags for velocity
26	UINT2	Threshold flags for width
28	UINT2	Threshold flags for ZDR
30	6	<spare>
36	UINT2	Flags: Bit 0: Speckle remover for log channel Bit 3: Speckle remover for linear channel Bit 4: Flag to indicate data is range normalized Bit 5: Flag to indicate pulse at beginning of ray Bit 6: Flag to indicate pulse at end of ray Bit 7: Vary number of pulses in dual PRF Bit 8: Use 3 lag processing in PP0 2 Bit 9: Apply velocity correction for ship motion Bit 10: Vc is unfolded Bit 11: Vc has fallspeed correction Bit 12: Zc has beam blockage correction Bit 13: Zc has Z-based attenuation correction Bit 14: Zc has target detection Bit 15: Vc has storm relative velocity correction
38	2	<spare>
40	SINT2	LDR bias in signed 1/100 dB
42	SINT2	ZDR bias in signed 1/16 dB
44	SINT2	NEXRAD point clutter threshold in 1/100 of dB
46	UINT2	NEXRAD point clutter bin skip in low 4 bits
48	SINT2	I0 cal value, horizontal pol, in 1/100 dBm
50	SINT2	I0 cal value, vertical pol, in 1/100 dBm
52	SINT2	Noise at calibration, horizontal pol, in 1/100 dBm
54	SINT2	Noise at calibration, vertical pol, in 1/100 dBm
56	SINT2	Radar constant, horizontal pol, in 1/100 dB
58	SINT2	Radar constant, vertical pol, in 1/100 dB
60	UINT2	Receiver bandwidth in kHz

Byte	Size	Contents
62	uint16_t	Flags2: Bit 0: Zc and ZDRc has DP attenuation correction Bit 1: Z and ZDR has DP attenuation correction
64	256	<spare>

4.3.51 task_configuration structure

Source: `iris_task.h`

<structure_header>12 Bytes. See structure_header structure (page 55).
<task_sched_info> 120 Bytes. See task_sched_info structure (page 63).
<task_dsp_info>320 Bytes. See task_dsp_info structure (page 58).
<task_calib_info> 320 Bytes. See task_calib_info structure (page 56).
<task_range_info>160 Bytes. See task_range_info structure (page 62).
<task_scan_info> 320 Bytes. See task_scan_info structure (page 62).
<task_misc_info> 320 Bytes. See task_misc_info structure (page 61).
<task_end_info>320 Bytes. See task_end_info structure (page 60).
Comments 720 Bytes

4.3.52 task_dsp_info structure

Source: `iris_task.h`

Byte	Size	Contents
0	UINT2	Major mode
2	UINT2	DSP type
4	24	<dsp_data_mask> Current Data type mask[1)]
28	24	<dsp_data_mask> Current Data type mask[1)]
If Batch Major mode:		
52	32	<task_dsp_mode_batch>
Else:		
52	32	<task_dsp_mode_other>
84	52	<spare>
136	SINT4	PRF in Hertz

Byte	Size	Contents
140	SINT4	Pulse width in 1/100 of microseconds
144	UINT2	Multi PRF mode flag: 0=1:1 1=2:3 2=3:4 3=4:5
146	SINT2	Dual PRF delay
148	UINT2	AGC feedback code
150	SINT2	Sample size
152	UINT2	Gain Control flag (0=fixed, 1=STC, 2=AGC)
154	char[12]	Name of file used for clutter filter
166	UINT1	Linear based filter number for first bin
167	UINT1	Log based filter number for first bin
168	SINT2	Attenuation in 1/10 dB applied in fixed gain mode
170	UINT2	Gas attenuation in 1/100000 dB/km for first 10000, then stepping in 1/10000 dB/km
172	UINT2	Flag nonzero means cluttermap used
174	UINT2	XMT phase sequence: 0:Fixed 1:Random 3:SZ8/64
176	UINT4	Mask used for to configure the ray header.
180	UINT2	Time series playback flags, see `OPTS_*` in `dsp.h`
182	2	`<spare>`
184	char[16]	Name of custom ray header
200	120	`<spare>`

1) See dsp_data_mask structure (page 29).

4.3.53 task_dsp_mode_batch structure

Source: `iris_task.h`

Byte	Size	Contents
0	UINT2	Low PRF in Hz
2	UINT2	Low PRF fraction part, scaled by 2**–16
4	SINT2	Low PRF sample size

Byte	Size	Contents
6	SINT2	Low PRF range averaging in bins
8	SINT2	Threshold for reflectivity unfolding in 1/100 of dB
10	SINT2	Threshold for velocity unfolding in 1/100 of dB
12	SINT2	Threshold for width unfolding in 1/100 of dB
14	18	`<spare>`

4.3.54 task_end_info structure

Source: `iris_task.h`

Byte	Size	Contents
0	SINT2	Task major number
2	SINT2	Task minor number
4	char[12]	Name of task configuration file
16	char[80]	Task description
96	SINT4	Number of tasks in hybrid task
100	UINT2	Task state: 0=no task 1=task being modified 2=inactive 3=scheduled 4=running
102	2	`<spare>`
104	12	`<ymds_time>` Data time of task (TZ flexible)
116	UINT1[6]	Identifiers of the echo classifiers in the HCLASS bit segments
122	198	`<spare>`

4.3.55 task_file_scan_info structure

Source: `iris_task.h`

Byte	Size	Contents
0	UINT2	First azimuth angle (binary angle)
2	UINT2	First elevation angle (binary angle)
4	char[12]	Filename for antenna control
16	184	`<spare>`

4.3.56 task_manual_scan_info structure

Source: `iris_task.h`

Byte	Size	Contents
0	UINT2	Flags: bit 0=Continuous recording
2	198	`<spare>`

4.3.57 task_misc_info structure

Source: `iris_task.h`

Byte	Size	Contents
0	SINT4	Wavelength in 1/100 of cm
4	char[16]	T/R Serial Number
20	SINT4	Transmit Power in watts
24	UINT2	Flags: Bit 0: Digital signal simulator in use Bit 1: Polarization in use Bit 4: Keep bit
26	UINT2	Type of polarization
28	SINT4	Truncation height (centimeters above the radar)
32	18	`<spare for polarization spec>`
50	12	`<spare>`
62	SINT2	Number of bytes of comments entered
64	BIN4	Horizontal beam width
68	BIN4	Vertical beam width
72	UINT4[10]	Customer defined storage
112	208	`<spare>`

4.3.58 task_ppi_scan_info structure

Source: `iris_task.h`

Byte	Size	Contents
0	BIN2	Left azimuth limit (binary angle, only for sector)
2	BIN2	Right azimuth limit (binary angle, only for sector)
4	UINT2[40]	List of elevations (binary angles) to scan at

84	115	<spare>
199	1	Start of first sector sweep: 0=Nearest 1=Left 2=Right Sector sweeps alternate in directions

4.3.59 task_range_info structure

Source: `iris_task.h`

Byte	Size	Contents
0	SINT4	Range of first bin in centimeters
4	SINT4	Range of last bin in centimeters
8	SINT2	Number of input bins
10	SINT2	Number of output range bins
12	SINT4	Step between input bins
16	SINT4	Step between output bins (in centimeters)
20	UINT2	Flag for variable range bin spacing (1=var, 0=fixed)
22	SINT2	Range bin averaging flag
24	136	<spare>

4.3.60 task_rhi_scan_info structure

Source: `iris_task.h`

Byte	Size	Contents
0	UINT2	Lower elevation limit (binary angle, only for sector)
2	UINT2	Upper elevation limit (binary angle, only for sector)
4	UINT2[40]	List of azimuths (binary angles) to scan at
84	115	<spare>
199	1	Start of first sector sweep: 0=Nearest 1=Lower 2=Upper Sector sweeps alternate in direction

4.3.61 task_scan_info structure

Source: `iris_task.h`

Byte	Size	Contents
0	UINT2	Antenna scan mode: 1: PPI sector 2: RHI 3: Manual 4: PPI cont 5: file
2	SINT2	Desired angular resolution in 1/1000°
4	2	`<spare>`
6	SINT2	Number of sweeps to perform
If RHI scan:		
8	200	`<task_rhi_scan_info>`
If PPI sector, or PPI continuous scan:		
8	200	`<task_ppi_scan_info>`
If File scan:		
8	200	`<task_file_scan_info>`
If PPI Manual or RHI Manual scan:		
8	200	`<task_manual_scan_info>`
In all cases:		
208	112	`<spare>`

4.3.62 task_sched_info structure

Source: `iris_task.h`

Byte	Size	Contents
0	SINT4	Start time (seconds within a day)
4	SINT4	Stop time (seconds within a day)
8	SINT4	Desired skip time (seconds)
12	SINT4	Time last run (seconds within a day) (0 for passive ingest)
16	SINT4	Time used on last run (seconds) (in file time to writeout)
20	SINT4	Relative day of last run (0 for passive ingest)

Byte	Size	Contents
24	UINT2	Flag: Bit 0 = ASAP Bit 1 = Mandatory Bit 2 = Late skip Bit 3 = Time used has been measured Bit 4 = Stop after running
26	94	<spare>

4.3.63 tdwr_psi_struct structure

Source: **headers.h**

Byte	Size	Contents
0	UINT4	Flags: bit0=LLWAS bit1=WARN bit2=SLINE
4	UINT4	Maximum range in cm
8	char[4]	Source ID
12	char[3]	Center field wind direction
15	UINT1	<spare>
16	char[2]	Center field wind speed
18	char[2]	Center field gust speed
20	UINT4	Mask of protected areas checked
24	UINT4	Warning count
28	UINT4	SLINE count
32	UINT4	Forecast time

4.3.64 tdwr_results structure

Source: **product.h**

Byte	Size	Contents
0	char[16]	Corridor name, null terminated
16	char[8]	Arena Name, null terminated
24	char[3]	Alert Type
27	1	<spare>

Byte	Size	Contents
28	char[3]	Threshold direction
31	1	<spare>
32	char[2]	Threshold speed
34	2	<spare>
36	SINT4	Alert speed loss/gain in mm/sec, negative=loss
40	UINT4	Mask of protected areas checked for this corridor
44	UINT4	Mask of protected areas hit in this corridor
48	52	<spare>

4.3.65 text_results structure

Source: `product.h`

Byte	Size	Contents
0	char[512]	null terminated string of arbitrary text to be spoken by IRIS

4.3.66 top_psi_struct structure

Source: `headers.h`

Byte	Size	Contents
0	uint32_t	Flags: Bit 0: For THICK product only, make pseudo thickness
4	int16_t	Z threshold in 1/16 dBZ

4.3.67 track_psi_struct structure

Source: `headers.h`

Byte	Size	Contents
0	SINT4	Centroid area threshold in square meters
4	SINT4	Threshold level for centroid
8	UINT4	Protected area mask
12	SINT4	Maximum forecast time in seconds
16	UINT4	Maximum age between products for motion calculation
20	SINT4	Maximum motion allowed in mm/sec

Byte	Size	Contents
24	UINT4	Flag word: Bit9=Generate diagnostic output
28	SINT4	Maximum span of track points in the file (seconds)
32	UINT4	Input product type
36	char[12]	Input product name
48	SINT4	Point connecting error allowance

4.3.68 track_results structure

Source: `product.h`

Byte	Size	Contents
0	BIN4	Latitude (32-bit binary angle)
4	BIN4	Longitude
8	SINT4	Height (cm above reference)
12	UINT4	Flags: bit0=forecast bit1=manual bit2=text bit3=icon
16	SINT4	Area of centroid (1/100 of square km)
20	SINT4	Major axis of equal area ellipse (cm)
24	SINT4	Minor axis of equal area ellipse (cm)
28	BIN4	Orientation angle of ellipse
32	UINT4	Protected area mask of areas hit
36	SINT4	Maximum value of data within area
40	8	`<spare>`
48	SINT4	Average value of data within area
52	8	`<spare>`
60	SINT4	Scale factor of input data
64	SINT4	Track index number
68	char[32]	Text
100	12	`<ymds_time>` Time (TZ flexible)
112	SINT4[32]	ETA in seconds for each protected area (0 if in area, -1 if not expected)
240	UINT4	Data type of input data
244	8	`<spare>`

Byte	Size	Contents
252	SINT4	Propagation speed (mm/second)
256	BIN4	Propagation direction (binary angle)
260	UINT4	Text size
264	UINT4	Color
268	32	<spare>

4.3.69 vad_psi_struct structure

Source: **headers.h**

Byte	Size	Contents
0	int32_t	Minimum slant range in cm.
4	int32_t	Maximum slant range in cm.
8	uint32_t	Flags: Bit 0: Unfold based on VVP product
12	uint32_t	Count of the number of elevation angles in the file

4.3.70 vad_results structure

Source: **headers.h**

Byte	Size	Contents
0	BIN2	Elevation angle
2	BIN2	Azimuth angle
4	uint16_t	Count of the number of bins averaged
6	uint16_t	Average velocity (DB_VEL2 format)
8	uint16_t	Standard deviation (cm/sec)
10	uint16_t	Elevation index, origin 0
12	8	<spare>

4.3.71 vvp_psi_struct structure

Source: **headers.h**

Byte	Size	Contents
0	SINT4	Minimum range to process in cm

Byte	Size	Contents
4	SINT4	Maximum range to process in cm
8	SINT4	Minimum height to process (cm above reference)
12	SINT4	Maximum height to process (cm above reference)
16	SINT2	Number of intervals to process at
18	uint16_t	Minimum velocity (cm/sec) (version 8.13 and later, previously 0.02 times nyquist)
20	SINT4	Quota number of bins per interval
24	SINT4	Wind parameters mask.See bits defined in vvp_results structure below.

4.3.72 vvp_results structure

Source: `product.h`

Byte	Size	Contents
0	SINT4	Number of data points used
4	SINT4	Height of the center of interval (cm above reference)
8	SINT4	Number of reflectivity data points used
12	8	`<spare>`
		Bit Description
20	SINT2	0 Wind speed in cm/sec
22	SINT2	1 Wind speed standard deviation
24	SINT2	2 Wind direction in 1/10 of degrees
26	SINT2	3 Wind direction standard deviation
28	SINT2	4 Vertical wind speed in cm/sec
30	SINT2	5 Vertical wind speed standard deviation
32	SINT2	6 Horizontal divergence in 10**–7/sec
34	SINT2	7 Horizontal divergence standard deviation
36	SINT2	8 Radial velocity standard deviation
38	SINT2	9 Linear averaged reflectivity (DB_CDBZ2 format)
40	SINT2	10 Log averaged reflectivity standard deviation
42	SINT2	11 Deformation in 10**–7/sec
44	SINT2	12 Deformation standard deviation
46	SINT2	13 Axis of dilatation in 1/10 of degrees
48	SINT2	14 Axis of dilatation standard deviation

Byte	Size	Contents
50	SINT2	15 Log averaged reflectivity (DB_CDBZ2 format)
52	SINT2	16 Linear averaged reflectivity standard deviation
54	30	<spare>

4.3.73 warn_psi_struct structure

Source: `headers.h`

Byte	Size	Contents
0	SINT4	Centroid area threshold in square meters
4	SINT4[3]	Threshold levels (1/100 of user units)
16	SINT2[3]	Data valid times (seconds)
22	2	<spare>
24	char[12]	Symbol to display
36	char[36]	Names of the product files
72	UINT1[3]	Product types
75	char[1]	<spare>
76	UINT4	Protected area bit flag

4.3.74 warning_results structure

Source: `product.h`

Byte	Size	Contents
0	BIN4	Latitude (32-bit binary angle)
4	BIN4	Longitude
8	SINT4	Height (cm above reference)
12	UINT4	Flags: Bit 0: Skip data label
16	SINT4	Area of centroid (1/100 of square km)
20	SINT4	Major axis of equal area ellipse (cm)
24	SINT4	Minor axis of equal area ellipse (cm)
28	BIN4	Orientation angle of ellipse
32	UINT4	Protected area mask of areas hit
36	SINT4[3]	Maximum value of data within the area (1/100 of user units)
48	SINT4[3]	Average value of data within the area (1/100 of user units)

Byte	Size	Contents
60	SINT4	Scale factor of input data
64	4	<spare>
68	char[16]	Text, null-terminated
84	156	<spare>
240	UINT4[3]	Data type of input data
252	SINT4	Propagation speed (mm/second)
256	BIN4	Propagation direction (binary angle)
260	40	<spare>

4.3.75 wind_psi_struct structure

Source: `headers.h`

Byte	Size	Contents
0	SINT4	Minimum height (cm above reference)
4	SINT4	Maximum height (cm above reference)
8	SINT4	Minimum range in cm
12	SINT4	Maximum range in cm
16	SINT4	Number of point in range
20	SINT4	Number of points in panels
24	SINT4	Sector length in cm
28	BIN4	Sector width in binary angle
32	UINT4	Flag word: bit0=Subtract mean wind
36	UINT4	Wind parameters mask of included VVP

4.3.76 wind_results structure

Source: `product.h`

Byte	Size	Contents
0	SINT4	Number of possible hits based on geometry
4	SINT4	Number of data points actually used
8	SINT4	Range of center of sector
12	BIN2	Azimuth of center of sector
14	SINT2	Velocity East in cm/second

Byte	Size	Contents
16	SINT2	Velocity East standard deviation in cm/second
18	SINT2	Velocity North in cm/second
20	SINT2	Velocity North standard deviation in cm/second
22	10	`<spare>`

4.3.77 ymds_time structure

Source: `sigtypes.h`

Byte	Size	Contents
0	SINT4	Seconds since midnight
4	UINT2	Milliseconds in lower 10 bits. Bit 10: Time is daylight savings time Bit 11: Time is UTC Bit 12: Local time is daylight savings time
6	SINT2	Year
8	SINT2	Month
10	SINT2	Day

4.4 Data types

For ingest data (stored in the raw product) and other products, the data formats for reflectivity, velocity, width, and ZDR differ slightly. When these data types are converted to a Cartesian product, the data values of 255 are replaced with 254, and 255 takes on the meaning of area not scanned.

For polar data, the name in parenthesis after each section title is the data type name used by IRIS to identify the data. Parameters for these numbers are defined in the *`sigtypes.h`* file.

Table 9 Data types

Data type	Description
`Ah, Av`	Integral attenuation for horizontal (H) and vertical (V) channels
`Azdr`	Integral attenuation of ZDR (dB) format
`CSR`	Doppler channel clutter-to-signal (CSR) ratio of dBT to -dBZ

Data type	Description
dBZ dBZv dBZe	Clutter-corrected reflectivity Vertical (HV enhanced) reflectivity Total reflectivity from the vertical polarization channel (**ZV**) and combination of the horizontal and vertical channel (**ZE**).Corrected for ground clutter.
dBT dBTv dBTe	Uncorrected reflectivity Total vertical (HV enhanced) reflectivity Total reflectivity from the vertical polarization channel (TV) and combination of the horizontal and vertical channel (TE)
HCLASS	Hydrometeor classification Estimated hydrometeor type in the precipitation area
KDP	Specific differential phase An indicator of the rate of change of the phase difference between horizontally and vertically polarized pulses of the radar. A greater horizontal shift results in a positive KDP value, and a greater vertical shift results in a negative KDP value. Typical cause for a high KDP area is heavy rain.
LDRH, LDRV	Linear depolarization ratio H to V (or V to H) The ratio of cross-polar to co-polar reflectivity measured in dB
LOG	Log receiver signal-to-noise ratio
PHIH, PHIV	Horizontal (V) or vertical (V) differential phase Phase difference for the total round trip between radar and the volume where the signal is reflected. **PHIH** is measured between HH and HV channels. **PHIV** is measured between VV and VH channels.
PHIDP	Differential phase The phase difference due to propagation between the HH and VV channels of the radar.
PMI	Polarimetric meteo index
R	Rate of accumulation of precipitation in units of mm/hour For snow, this usually refers to the liquid equivalent.
RHOHV, RHOH, RHOV	Correlation coefficient between HH and VV (or HH & HV / VV & VH) channels Higher (>0.95) values indicate uniform precipitation areas, and lower values indicate more mixed hydrometeor types, such as melting snow, wet snowflakes, or airborne debris.
SNR	Signal-to-noise ratio Generic measurement of signal-noise ratio in dB
SQI	Signal quality index A value between 0-1 that measures the signal's Doppler coherency, that is, the correlation between the signal and its Doppler lag: • 0 indicates white noise • 1 is the perfect Doppler point target
TV, TE	Total vertical (HV enhanced) reflectivity Total reflectivity from the vertical polarization channel (TV) and combination of the horizontal and vertical channel (TE)

Data type	Description
V	Velocity Average radial velocity (towards or away from the radar) of detected hydrometeor areas
VC	Corrected velocity Same as velocity V, but corrected for effects of range folding and velocity folding
V: SHEAR, Vc: SHEAR	Velocity and corrected velocity of wind shear
W	Spectral width Variability of Doppler velocity values within the measurement area
XCOR	Polar cross-correlation, uncorrected rhohv Because this value is not noise corrected, this is a direct indicator of the PHIDP uncertainty.
ZV, ZE	Vertical (HV enhanced) reflectivity Total reflectivity from the vertical polarization channel (ZV) and combination of the horizontal and vertical channel (ZE). Corrected for ground clutter.
ZC	Corrected reflectivity Same as reflectivity Z, but corrected for attenuation and beam blockage effects
ZDR	Differential reflectivity The ratio of SNR in the horizontal channel to the SNR in the vertical channel. Positive values indicate more prominent horizontal echoes and negative values more prominent vertical echoes. Larger hydrometeor sizes are usually identified by high positive ZDR values.
ZDRC	Corrected differential reflectivity Same as differential reflectivity ZDR, but corrected for attenuation and beam blockage effects

4.4.1 Extended header format (DB_XHDR)

The extended header is optionally recorded, as controlled by a question in **Setup**. Choose from either:

- `extended_header_v0`
- `extended_header_v1`

`extended_header_v1` is usually for moving-platform systems, such as ships.

4.4.2 2-byte axis of dilation format (DB_AXDIL2)

The angle in degrees is stored as a signed number in tenths of degrees.

$$\text{angle} = \frac{N}{10}$$

-1800	-180.0°
0	0.0°
10	1.0°

4.4.3 1-byte AZDR format (DB_AZDR8)

1-byte integral attenuation of ZDR (dB) format.

This attenuation moment is computed even if dual-pol attenuation correction is not requested. You can use this moment for corrections during post-processing, to develop new quality control algorithms, or to improve rain accumulation estimates.

Same format as **DB_SNR8**, see 1-byte SNR format (DB_SNR8) (page 85).

4.4.4 2-byte AZDR format (DB_AZDR16)

1-byte integral attenuation of ZDR (dB) format.

This attenuation moment is computed even if dual-pol attenuation correction is not requested. You can use this moment for corrections during post-processing, to develop new quality control algorithms, or to improve rain accumulation estimates.

Same format as **DB_SNR16**, see 2-byte SNR format (DB_SNR16) (page 85).

4.4.5 1-byte Clutter Signal Power format (DB_CSP8)

1-byte Doppler channel clutter power ratio of dBT to -dBZ.

Same format as **DB_SNR8**, see 1-byte SNR format (DB_SNR8) (page 85).

4.4.6 2-byte Clutter Signal Power (DB_CSP16)

1-byte Doppler channel clutter power ratio of dBT to -dBZ.

Same format as **DB_SNR16**, see 2-byte SNR format (DB_SNR16) (page 85).

4.4.7 1-byte CSR format (DB_CSR8)

1-byte Doppler channel clutter-to-signal ratio (CSR). Same format as **DB_SNR8**, see 1-byte SNR format (DB_SNR8) (page 85).

4.4.8 2-byte CSR format (DB_CSR16)

1-byte Doppler channel clutter-to-signal ratio (CSR). Same format as **DB_SNR16**, see 2-byte SNR format (DB_SNR16) (page 85).

4.4.9 2-byte deformation format (DB_DEFORM2)

Deformation is stored in a signed 16-bits number scaled to 10**-7. Only positive numbers are possible. These number are normally displayed in units of 10**-4.

0	0 deformation
1	0.001 10**-4 deformation
32766	32.766 10**-4 deformation
32767	Area not scanned

4.4.10 2-byte divergence format (DB_DIVERGE2)

Divergence is stored in a signed 16-bits number scaled to 10**-7. Positive numbers indicate divergence, negative are convergence. These number are normally displayed in units of 10**-4.

-32768	32.768 10**-4 convergence
0	0 divergence
1	0.001 10**-4 divergence
32766	32.766 10**-4 divergence
32767	Area not scanned

4.4.11 1-byte echo tops format (DB_HEIGHT)

Echo tops are stored to the nearest 100 meters above the ground. Because 0 is the code for no data, values stored in the bytes can be converted to kilometers by subtracting 1 and dividing by 10.

The value 254 indicates that an echo is known to exist from the data, but it could not be measured.

0	No echo tops data available
1	0.0 km
128	12.7 km
129	12.8 km
253	25.2 km or above
254	Top exists above the maximum tilt
255	Area not scanned

4.4.12 2-byte floating liquid format (DB_FLIQUID2)

Rainfall accumulations are stored in 16-bit floating-point representation of a 27-bit integer in units of 0.001 mm.

This product does not have a code for thresholded data, instead that area is assumed to have zero rain.

The floating-point number consists of a 4-bit exponent in the high four bits, followed by a 12-bit mantissa. It uses implied digits and soft underflow. If the exponent is 0, the mantissa consists of the value. If the exponent is 1 through 15, the 12-bit mantissa has a 1 prefixed to it and is shifted up by one less than the exponent. The table below gives some examples:

Float		Fixed	Meaning
0	—>	0	0.000 mm
1	—>	1	0.001 mm
255	—>	255	0.255 mm
1000	—>	1000	1.000 mm
9096	—>	10000	10.000 mm
22634	—>	100000	100.000 mm
34922	—>	800000	800.000 mm
50000	—>	10125312	10125.312 mm
65534	—>	134184960	134184.960 mm or above
65535	—>		Area not scanned

4.4.13 2-byte horizontal wind direction format (DB_HDIR2)

The angle in degrees is stored as a signed number in tenths of degrees.

$$\text{angle} = \frac{N}{10}$$

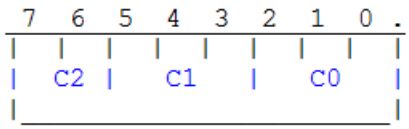
-1800	-180.0°
0	0.0°
10	1.0°

4.4.14 1-byte HydroClass format (DB_HCLASS)

HydroClass consists of enumerated data for communicating echo identification results. The 1-byte data type can accommodate results from up to 3 independent identification methods or classification hypotheses. The following decimal values are reserved:

- `0` (0x00) for `No data`
- `255` (0x255) `Area not scanned`

1-byte of binary DB_HCLASS data is split into 3 bit segments as follows:



Each segment can be associated with an identification method (echo classifier). For a list of the available classification methods listed as enumerated logical names, see the public IRIS/RDA header *sig_data_types.h*.

The bytes **Enums** of the **task_dsp_info** structure (see task_dsp_info structure (page 58)) contain the list of classification methods allocated to each bit segment of the HClass data. The information is associated to each **IRIS RAW** data file header, and can be inspected with the **productx** utility.

The bits in each segment are interpreted as an unsigned integer value belonging to an enumerated class type (the class base), specific to each classifier. The enumerated class types are defined in *sig_data_types.h* with unambiguous logical names that are meteorologically explicit.

The default configuration of IRIS/RDA delivers the echo classification results shown in the following tables. You can modify the configuration using settings in *hydroclass-*-band.conf*).



The methods **METEO** and **PRECIPCLASSIFIER** are available both in RDA real-time ray level processing, as well as in IRIS reprocessing (reingest).
CELLCLASSIFIER is available in IRIS post-processing only. Its outcomes can be interpreted in the context of ray level classifications.
Further classification information can be appended to 2-byte HydroClass data. See 2-byte HydroClass format (DB_HCLASS2) (page 78).

Table 10 Method 1 - METEOCLASSIFIER (HydroClass)

MET_CLASS_THRESHOLD	=0 No data available
MET_CLASS_NON_MET	=1 Non-meteorological target
MET_CLASS_RAIN	=2 Rain
MET_CLASS_WET_SNOW	=3 Wet Snow
MET_CLASS_SNOW	=4 Snow
MET_CLASS_GRAUPEL	=5 Graupel
MET_CLASS_HAIL	=6 Hail
	=7 Unused

Table 11 Method 2 - PRECIPCLASSIFIER

PRE_CLASS_THRESHOLD	=0 No data available

PRE_CLASS_GC_AP	=1 Ground clutter / anomalous propagation
PRE_CLASS_BIO	=2 Bio scatter
PRE_CLASS_PRECIP	=3 Precipitation
PRE_CLASS_LARGE_DROPS	=4 Large drops
PRE_CLASS_LIGHT_PRECIP	=5 Light precipitation
PRE_CLASS_MODERATE_PRECIP	=6 Moderate precipitation
PRE_CLASS_HEAVY_PRECIP	=7 Unused



For **PRECIPCLASSIFIER**, the interpretations into classes from **Large Drops** to **Heavy precipitation** are available if **HClass PrecipClassifier** is **ON**.
Without **HClass PrecipClassifier**, all precipitation are classified as **Precipitation**.

Table 12 Method 3 - CELLCLASSIFIER

CELL_CLASS_STRATIFORM	=0 Stratiform
CELL_CLASS_CONVECTION	=1 Convection
	=2 Unused
	=3 Forbidden

4.4.15 2-byte HydroClass format (DB_HCLASS2)

The interpretation of the 8 least significant bits is the same as the 1-byte format, except that 65535 (**0xFFFF**) is used for **area not scanned**.

The currently available identification methods (as of IRIS/RDA 8.12.6) can be accommodated in the 8 least significant bits of **DB_HCLASS2**, see 1-byte HydroClass format (DB_HCLASS) (page 76).

As default, the remaining 8 most significant bits of **DB_HCLASS2** replicate the same formatting as the lower 1-byte bits, for support of three further identification methods. They are reserved for future uses, such as re-analysis of precipitation type at surface.

It is also possible to merge the bit segments by combining the segments within the 8 most significant bits, multiplicatively. In that scenario, the **DB_HCLASS2** data type can support a user specified identification method in large bases up to 7x7x3= 147 classes.

4.4.16 1-byte integral attenuation (DB_AH8) and (DB_AV8)

1-byte integral attenuation for horizontal (**H**) and vertical (**V**) channels.

This attenuation moment is computed even if dual-pol attenuation correction is not requested. You can use this moment for corrections during post-processing, to develop new quality control algorithms, or to improve rain accumulation estimates.

Same format as **DB_SNR8**, see 1-byte SNR format (DB_SNR8) (page 85).

4.4.17 2-byte integral attenuation (DB_AH16) and (DB_AV16)

2-byte integral attenuation for horizontal (**H**) and vertical (**V**) channels.

This attenuation moment is computed even if dual-pol attenuation correction is not requested. You can use this moment for corrections during post-processing, to develop new quality control algorithms, or to improve rain accumulation estimates.

Same format as **DB_SNR16**, see 2-byte SNR format (DB_SNR16) (page 85).

4.4.18 1-byte KDP format (DB_KDP)

Specific differential phase (KDP) is stored in a signed 8-bit number using a log scale. The KDP angles are multiplied by the wavelength in cm then scaled using a log scale separately for both signs. The minimum value is 0.25, and the maximum value is 150.0 deg*cm/km. KDP values are defined as the one-way differential effect of the intervening weather. The table below gives some examples:

$$\mathrm{KDP} \times \lambda = \mathrm{minimum} \times \left[\frac{\mathrm{minimum}}{\mathrm{maximum}}\right]^{\left[\frac{\mathrm{N}-129}{126}\right]}$$

Here is the conversion equation for positive values (stored value above 128):

$$\mathrm{KDP} \times \lambda = 0.25 \times [600]^{\left[\frac{\mathrm{N}-129}{126}\right]}$$

Here is the conversion equation for negative values (stored value below 128):

$$\mathrm{KDP} \times \lambda = 0.25 \times [600]^{\left[\frac{127-N}{126}\right]}$$

Value	Meaning KDP*L	KDP for 10 cm	KDP for 5 cm
0	No data available		
1	-150.00 deg*cm/km	-15.00 deg/km	-30.00 deg/km
2	-142.58	-14.26	-28.51

Value	Meaning KDP*L	KDP for 10 cm	KDP for 5 cm
127	-0.250	-0.025	-0.050
128	0.000	0.000	0.000
129	0.250	0.025	0.050
130	0.263	0.026	0.053
254	142.58	14.58	28.51
255	Area not scanned		

4.4.19 2-byte KDP format (DB_KDP2)

Specific differential phase (KDP) in degrees per kilometer is computed from the unsigned output with:

$$\mathrm{KDP} = \left[\frac{N - 32768}{100}\right]$$

The overall range is -327.67 ... +327.66 in 1/100 of a degree/kilometer steps as follows.

0	No data available
1	-327.67 deg/km
32768	0.00 deg/km
32769	0.01 deg/km
65534	327.66 deg/km
65535	Reserved for area not scanned in product files

4.4.20 1-byte LDR format (DB_LDRH & DB_LDRV)

LDR (Linear Depolarization Ratio) is the ratio of the power received in the depolarized channel to the power received in the main channel:

- When transmitting horizontally, the ratio of the vertical power to the horizontal power is LDRH.
- When transmitting vertically, the ratio of horizontal power to vertical power is LDRV.

In simultaneous transmission LDR cannot be computed because the signal in the depolarized channel is dominated by co-polarized returns.

$$LDR\left(db\right) = \frac{\mathrm{N} - 1}{5} - 45.0$$

The overall range (in dB) is -45 ...+5.6 in steps of 0.2 dB as follows.

0	No data available
1	-45.0 dB
2	-44.8 dB
226	0.0 dB
254	+5.6 dB
255	Reserved for area not scanned in product files

4.4.21 2-byte LDR format (DB_LDRH2 & DB_LDRV2)

Same as DB_DBZ2 format, see 2-byte reflectivity format (DB_DBT2& DB_DBZ2) (page 84).

4.4.22 1-byte LOG format (DB_LOG8)

1-byte log receiver signal-to-noise ratio (dB). Same format as **DB_SNR8**, see 1-byte SNR format (DB_SNR8) (page 85).

4.4.23 2-byte LOG format (DB_LOG16)

1-byte log receiver signal-to-noise ratio (dB). Same format as **DB_SNR16**, see 2-byte SNR format (DB_SNR16) (page 85).

4.4.24 1-byte Phi format (DB_PHIH & DB_PHIV)

The cross channel differential phase. Same format as DB_PHIDP, see 1-byte PhiDP format (DB_PHIDP) (page 81).

4.4.25 2-byte Phi format (DB_PHIH2 & DB_PHIV2)

The cross channel differential phase. Same format as DB_PHIDP2, see 2-byte PhiDP format (DB_PHIDP2) (page 82).

4.4.26 1-byte PhiDP format (DB_PHIDP)

PhiDP is the two-way measured phase effect after the signal has travelled through the intervening weather twice. Differential phase (PhiDP) in degrees is computed from the unsigned output with:

$$\Phi_{DP} \bmod 180 = 180 \times \frac{\mathrm{N} - 1}{254}$$

The overall range is from 0 ... 180° in steps of 0.71 as follows.

0	No data available
1	0.00 deg
2	0.71 deg
101	70.87 deg
254	179.29 deg
255	Reserved for area not scanned in product files

4.4.27 2-byte PhiDP format (DB_PHIDP2)

PhiDP is defined as the 2-way measured phase effect after the signal has travelled through the intervening weather twice. Differential phase (PhiDP) in degrees is computed from the unsigned output with:

$$\Phi_{DP} \bmod 360 = 360 \times \frac{\mathrm{N} - 1}{65534}$$

The overall range is 0 ... 360° degrees in steps of 0.0055 as follows. In cases where the transmitter was alternating polarization, or the data was converted from 1–byte format, the values only cover 0 ... 180°.

0	No data available
1	0.0000 deg
2	0.0055 deg
65534	359.9945 deg
65535	Reserved for area not scanned in product files

4.4.28 1-byte PMI format (DB_PMI8)

1-byte Polarimetric Meteo Index (PMI) format, which is unitless and spans 0 ... 1. Same format as **DB_SQI**, see 1-byte Signal Quality Index format (DB_SQI) (page 85).

4.4.29 2-byte PMI format (DB_PMI16)

2-byte Polarimetric Meteo Index (PMI) format, which is unitless and spans 0 ... 1. Same format as **DB_SQI2**, see 2-byte Signal Quality Index format (DB_SQI2) (page 86).

4.4.30 2-byte Rainfall rate format (DB_RAINRATE2)

Rainfall rates are stored in 16-bit floating-point representation of a 27-bit integer in units of 0.0001 mm/hr.

The floating-point number consists of a 4-bit exponent in the high four bits, followed by a 12-bit mantissa. It uses implied digits and soft underflow. If the exponent is 0, the mantissa consists of the value.

If the exponent is 1 ... 15, the 12-bit mantissa has a 1 prefixed to it, making it a 13-bit mantissa. This 13-bit mantissa is shifted up by one less than the exponent.

The following table gives some examples.

Float		Fixed	Meaning
0	—>	0	No data available
1	—>	1	0.0000 mm/hr
2	—>	2	0.0001 mm/hr
225	—>	225	0.0254 mm/hr
1000	—>	1000	0.0999 mm/hr
9096	—>	10000	0.9999 mm/hr
22634	—>	100000	9.9999 mm/hr
34922	—>	800000	79.9999 mm/hr
50000	—>	10125312	1012.5311 mm/hr
65534	—>	134184960	13418.4959 mm/hr or above
65535	—>		Area not scanned

4.4.31 1-byte reflectivity format (DB_DBT& DB_DBZ)

For reflectivity data, the number in decibels is computed from the unsigned output with the formula:

$$dBZ = \frac{\mathrm{N} - 64}{2}$$

The overall range is from -31.5 ... +95.5 dBZ in 0.5 dB steps as follows. In data product files, the value of 255 indicates areas not scanned.

	Ingest data	Products
0	No data available	No data available
1	-31.5 dBZ	-31.5 dBZ
64	0.0 dBZ	0.0 dBZ
128	32.0 dBZ	32.0 dBZ
129	32.5 dBZ	32.5 dBZ

	Ingest data	Products
254	95.0 dBZ	95.0 dBZ or above
255	95.5 dBZ or above	Area not scanned

4.4.32 2-byte reflectivity format (DB_DBT2& DB_DBZ2)

For reflectivity data, the number in decibels is computed from the unsigned output with:

$$dBZ = \frac{\mathrm{N} - 32768}{100}$$

The overall range is -327.67 ... +327.66 in 1/100 of a dB steps as follows.

0	No data available
1	-327.67 dBZ
32768	0.00 dBZ
32769	0.01 dBZ
65534	327.66 dBZ
65535	Reserved for area not scanned in product files

4.4.33 1-byte Rho format (DB_RHOH & DB_RHOV)

The cross channel correlation coefficient. Same format as DB_RHOHV, see 1-byte RhoHV format (DB_RHOHV) (page 84).

4.4.34 2-byte Rho format (DB_RHOH2 & DB_RHOV2)

The cross channel correlation coefficient. Same format as DB_SQI2, see 2-byte Signal Quality Index format (DB_SQI2) (page 86).

4.4.35 1-byte RhoHV format (DB_RHOHV)

RhoHV is a measure of how correlated the power fluctuations in reflectivity in the horizontal receiver is to the vertical receiver. It is a dimensionless number on a scale for which 0 means no correlation, and 1 means complete correlation. Since numbers are more likely near 1, the data is scaled with a square root.

$$RhoHV = \sqrt{\frac{\mathrm{N} - 1}{253}}$$

0	Data not available at this range
1	0.0000
2	0.0629
128	0.7085
253	0.9980
254	1.0000
255	Area not scanned

4.4.36 2-byte RhoHV format (DB_RHOHV2)

Same format as DB_SQI2, see 2-byte Signal Quality Index format (DB_SQI2) (page 86).

4.4.37 1-byte SNR format (DB_SNR8)

1-byte signal-to-noise ration (SNR) format.

The overall range is from +32.5 ... +127.5 dB in 0.5 dB steps. In data product files, the value of 255 indicates areas not scanned.

4.4.38 2-byte SNR format (DB_SNR16)

2-byte signal-to-noise ration (SNR) format.

The overall range is from -327.67 ... +327.66 dB in 0.01 dB steps. In data product files, the value of 255 indicates areas not scanned.

4.4.39 1-byte Signal Quality Index format (DB_SQI)

The Signal Quality Index (SQI) is the ratio of the magnitude of R1 to the magnitude of R0. It is a dimensionless number on a scale for which 0 means pure noise, and 1 means a pure sine wave.

$$SQI = \sqrt{\frac{\mathrm{N}-1}{253}}$$

0	SQI data not available at this range
1	0.0000
2	0.0629
128	0.7085
253	0.9980
254	1.0000

255	Area not scanned

4.4.40 2-byte Signal Quality Index format (DB_SQI2)

Stored linearly using 16-bits as follows:

$$SQI = \frac{(N-1)}{65533}$$

0	SQI data not available at this range
1	0.00000
2	0.00002
128	0.00194
65533	0.99998
65534	1.00000
65535	Area not scanned

4.4.41 2-byte time format (DB_TIME2)

Stored as time in seconds, so in minutes the conversion is as follows:

$$Time = \frac{N - 32768}{60}$$

0	Time data not available at this range
1	- 9:06:07:
32768	00:00:00
32828	00:01:00
65535	Area not scanned

4.4.42 1-byte unfolded velocity format (DB_VELC)

This is the mean radial velocity corrected for both Nyquist folding and fall speed. It is scaled to cover a fixed span of +/- 75 meters/second. In data product files, the value 255 is used to indicate area not scanned.

0	Velocity data not available at this range
1	-75.0 m/s (towards the radar)

2	-74.4 m/s
128	Zero velocity
129	+0.6 m/s
254	+75.0 m/s (away from the radar)
255	Area not scanned

4.4.43 2-byte unfolded velocity format (DB_VELC2)

Mean velocity in meters per second is computed from the unsigned output with

$$velocity = \frac{N - 32768}{100}$$

The overall range is -327.67 ... +327.66 m/s in 1/100 of a meter per second steps as follows.

0	No data available
1	-327.67 m/s (towards the radar)
32768	0.00 m/s
32769	0.01 m/s
65534	327.66 m/s (away from the radar)
65535	Reserved for area not scanned in product files

4.4.44 1-byte velocity format (DB_VEL)

This is the mean velocity with respect to the Nyquist velocity.

In data product files, the value 255 is used to indicate area not scanned. The Nyquist velocity is wavelength times PRF divided by 4. For 2:3 dual PRF mode, it is doubled; for 4:3 PRF mode, it is tripled, and for 4:5 PRF mode it is quadrupled over that of the higher PRF. For alternating polarization it is halved.

$$velocity = \frac{N - 128}{127} \times Nyquist$$

0	Velocity data not available at this range
1	Unambiguous velocity towards the radar
128	Zero velocity
255	Unambiguous velocity away from the radar

4.4.45 2-byte velocity format (DB_VEL2)

Mean velocity in meters per second is computed from the unsigned output with

$$velocity = \frac{N - 32768}{100}$$

The overall range is -327.67 ... +327.66 m/s in 1/100 of a meter per second steps as follows.

0	No data available
1	-327.67 m/s (towards the radar)
32768	0.00 m/s
32769	0.01 m/s
65534	327.66 m/s (away from the radar)
65535	Reserved for area not scanned in product files

4.4.46 2-byte vertical velocity format (DB_VVEL2)

Vertical velocity is stored in a signed 16-bits number scaled to 0.01 meters/second. Positive number indicate upward motion, negative downward.

-32768	327.68 m/s upward motion
0	0 motion
1	0.01 m/s fall speed
32766	327.66 m/s fall speed
32767	Area not scanned

4.4.47 2-byte VIL format (DB_VIL2)

Vertically integrated liquid is stored in 16-bits to the nearest 0.001 mm. Because 0 is the code for no data, values stored in the bytes can be converted to millimeters by subtracting 1 and dividing by 1000.

0	No VIL data available
1	0.000 mm
128	0.127 mm
129	0.128 mm
255	0.254 mm
65534	65.533 mm or above

65535	Area not scanned

4.4.48 1-byte width format (DB_WIDTH)

Spectrum width is computed from the unsigned output as:

$$W = \frac{N}{256}$$

The overall range is a fraction between 1/256 ... 255/256.

The code of 0 indicates that width data is not available at this range.

To convert the width to m/s, multiply by the unambiguous velocity. Thus the width has twice the resolution of the velocity.

This unambiguous velocity is not enlarged by the dual PRF scheme, but is halved by alternating polarization.

In data products, the value 255 indicates area not scanned. Note that width unambiguous velocities are not changed for dual PRF unfolding.

4.4.49 2-byte width format (DB_WIDTH2)

Spectral width in meters per second is computed from the unsigned output with

$$width = \frac{N}{100}$$

The overall range is 0.01 ... 655.34 in 1/100 of a meter per second steps as follows.

0	No data available
1	0.01 m/s
32768	327.68 m/s
32769	327.69 m/s
65534	655.34 m/s
65535	Reserved for area not scanned in product files

4.4.50 1-byte wind shear format (DB_SHEAR)

Wind shear is stored to the nearest 0.2 meters per second per kilometer. This is a signed number, with positive indicating wind is increasing in velocity away from the radar with increasing range.

To convert, subtract 128 and multiply by 0.2.

0	No wind shear data available
1	-25.4 m/s/km or above
128	0.0 m/s/km
129	+0.2 m/s/km
254	+25.2 m/s/km or above
255	Area not scanned

4.4.51 1-byte XCOR format (DB_XCOR8)

Polar cross-correlation, uncorrected **rhohv**. Because this value is not noise corrected, it is a direct indicator of the **PHIDP** uncertainty

Same format as **DB_SQI**, see 1-byte Signal Quality Index format (DB_SQI) (page 85).

4.4.52 2-byte XCOR format (DB_XCOR8)

Polar cross-correlation, uncorrected **rhohv**. Because this value is not noise corrected, it is a direct indicator of the PHIDP uncertainty

Same format as **DB_SQI2**, see 2-byte Signal Quality Index format (DB_SQI2) (page 86).

4.4.53 1-byte ZDR format (DB_ZDR)

For differential reflectivity data, the number in decibels is computed from the unsigned output with the formula:

$$dB\left(ZDR\right) = \frac{N - 128}{16}$$

The overall range is -7.94 dBZ ... +7.94 dBZ in sixteenth of a dB steps as shown below.

Positive ZDR means that the horizontal return is stronger than the vertical return. In data products, the value 255 indicates area not scanned.

0	No ZDR data available
1	-7.94 dB
128	0.00 dB
129	+0.06 dB
255	+7.94 dB

4.4.54 2-byte ZDR format (DB_ZDR2)

For differential reflectivity data, the number in decibels is computed from the unsigned output with

$$dB\left(ZDR\right) = \frac{N - 32768}{100}$$

The overall range is -327.67 ... +327.66 in 1/100 of a dB steps as follows.

0	No data available
1	-327.67 dB
32768	0.00 dB
32769	0.01 dB
65534	327.66 dB
65535	Reserved for area not scanned in product files

4.5 Ingest data file format

Each ingest data file contains one data type from one sweep of a volume.

If extended headers are recorded, they are treated as another data type and are placed in a separate file. A the following table shows, each of these files consists of:

- fixed `length ingest_data_header` structure
- variable-length ray pointer table
- variable-length segment of data

The ray pointer table contains a 32-bit pointer for each ray in the file. These are byte pointers (origin one) referenced to the beginning of the data area. A pointer value of 1 refers to the first byte of the data area. Ray data are forced onto 16-bit word boundaries, so all the pointers are odd. A pointer value of 0 indicates that no data are available for that slot.

Table 13 Ingest data file format

`<ingest_data_header>` 76 Bytes
Pointer Table4 Bytes per expected ray
Data Area Undetermined Size

Ray data are partially compressed with a truncation scheme. If a ray does not contain data all the way out to the last range bin, the trailing bins are removed from the archive.

This means that the storage space required for each ray varies, which requires the pointer table for quick random access. The pointer points to a `ray_header` structure, which is followed by an array of range bins. The count of the number of range bins could be zero, in which case the ray contains only the 12-byte ray header.

4.5.1 Ingest file names

IRIS ingest data for a volume scan is stored on disk in multiple files.

The ingest summary file name is the prefix with a trailing ".", but without the suffix.

Each sweep and data type is stored in a separate file.

These files must be named correctly, **siris** deletes any with an incorrect name. If the two-digit year is less than 50, it means add 2000, otherwise add 1900.

Format	Example
`SSSYYMMDDHHMMSS.##DD`	`SIG010620141921.01dBZ`
Where:	
`SSS`—Three letter site code from Setup	`SIG`—Abbreviation for SIGMET
`YYMMDDHHMMSS`—Data time	`140620141921`—14:19:21 20 June 2014
`##`—Two-digit sweep number	`01`—Sweep #1
`DD`—data type (1–5 chars)	`dBZ`—Reflectivity

4.6 Product file format

Each product is stored in a separate file in the directory specified by the `IRIS_PRODUCT` environment variable.

Raw products are stored in a separate directory specified by the `IRIS_PRODUCT_RAW` environment variable.

Product file names consist of the 15-character site and time to the left of the dot, and a 7-character product type code and machine generated string to the right.

The file consists of the `product_hdr` structure followed by the data.

The product configuration structure is exactly what is in one of the product configuration files. It specifies how a product should be generated, and so includes some information not strictly required to appear in the final disk file.

Table 14 Product file format

`<product_hdr>` 640 Bytes
Optional `<protect_setup>` 1024 Bytes

Binary Data Area Variable Size
Optional Extended Product Header Variable Size
Optional Additional Binary Data Area Variable Size

4.6.1 Product file names

IRIS can handle a product with any file name up to 23 characters.

When IRIS creates a product file in its own product directory, it uses the following file name syntax.

The 2 digit year used is the year modulo 100. Since the product file names are never parsed to generate a full date, there is no need to ever reconstruct the correct century.

General format	Example
SSSYYMMDDHHMMSS.PPPXXXX	*SIG940620141921.TRAE090*
Where:	
SSS—Three letter site code from Setup	*SIG*—Abbreviation for SIGMET
YYMMDDHHMMSS—Data time	*940620141921*—14:19:21 20 June 1994
PPP—Three-letter product type	*TRA*—Track product
XXXX—Characters for uniqueness	*E090*

When IRIS copies files to other directories using the network product output, it must generate a file name that is unique in the target directory. For more information, see *SETUP/OUTPUT* in *IRIS Utilities Manual*.

4.6.2 Cartesian product format

The data portion of Cartesian product files is not compressed and consists of an array.

The first byte contains the lower left corner of the image, the second contains the pixels to the right of that, and so on from the bottom of the window to the top. For 3-D products, the lowest 2-D image comes first.

In some cases a product header extension is added to the Cartesian products, and may store multiple data types. See product_header_extensions (page 44).

4.6.3 FCAST product format

The FCAST product format is the same as other Cartesian products except that the data elements consist of the **`ndop_results`** structure, rather than a 1- or 2-byte number.

4.6.4 MLHGT product format

The MLHGT product format is the same as other Cartesian products except that the data elements also include the uncertainties.

Additional information about the melting layer classifier and the product are included in the header.

4.6.5 NDOP product format

The NDOP product format is the same as other Cartesian products except that the data elements consist of the ndop_results structure, rather than a 1- or 2-byte number. 3-D data is supported.

4.6.6 RAW product format

The RAW product is a collection of all raw ingest data acquired during a run of a single task (volume scan).

The raw product is a single file into which many ingest files have been incorporated.

For hybrid tasks, individual raw products are made from each of the individual tasks that make up the overall scan. Optionally the RAW product can be made in separate files on a sweep-by-sweep basis.

Raw product files are blocked into 6144-byte records, which match the record length used if and when the files are eventually written to tape. Mimicking the tape structure on disk permits error recovery to be built directly into the raw data format. For all other types of products, if a tape I/O error occurs within the product's records on tape, then the entire product is lost. For raw archive this would be too great a penalty, since an entire volume scan is at stake. The blocking scheme permits partial error recovery while still maintaining a one-to-one mapping between disk and tape formats.



The records are for IRIS interpretation only and do not refer to any operating system file records.

The first 6144 byte record of a RAW product holds the **`product_hdr`** (See product_hdr structure (page 43)) structure.

The RAW product begins with a **`product_hdr`** like all the other types of products. The only difference is the zero padding out to 6144 bytes.

The next record holds the **`ingest_header`** (see ingest_header structure (page 34)) structure for the volume scan that supplied the data. This structure is zero-padded to fill the entire second record.

All subsequent records hold the actual data, and each record begins with the **`raw_prod_bhdr`** structure. See 1-byte velocity format (DB_VEL) (page 87).

Records hold data from one sweep. The sweep number is in each record's header. The data for the sweep is the concatenation of all the data from as many records as pertain to that sweep. When data for a sweep ends short of the 6144 byte record size, the remainder of that record is padded with zeros. Each of these sweep data sets begins with the **ingest_data_header** (See ingest_data_header structure (page 33)) structures for each data type that was recorded. The same number of headers are found in the beginning of a sweep's data as there are data types acquired during the sweep. The list of data types recorded is specified by the data collection mask in the **task_dsp_info** in the **task_configuration** in the **ingest_header**. The rays of data are immediately after the headers.

Rays are ordered within a sweep by the same ordering sequence used for the ingest data file pointer table. If a ray is missing from the ingest file, a zero-length ray is inserted into the product file as a placeholder. The number of compressed rays in the file is equal to the product of the number of data types recorded and the number of angles sampled. This is true even if data were not acquired at some of those angles. Within a ray, the recorded data are ordered by increasing data type number. See the **task_dsp_info** structure for a definition of the data type numbers. See task_dsp_info structure (page 58).

Data Compression

The raw product headers and the ingest data headers are not compressed. The overall organization of the file is shown below. Raw product files are blocked into 6144-byte records, and all but the first 2 records begin with the **raw_prod_bhdr** 12-byte structure.

For information on the algorithm used to compress data rays, see Data compression algorithm (page 95).

```
Record #1 { <product_hdr> 0,0,0... }
Record #2 { <ingest_header> 0,0,0... }
Record #3 { <raw_prod_bhdr> <ingest_data_header(s)> Data... }
Record #4 { <raw_prod_bhdr> Data... }
. . .
. . .
Record #N { <raw_prod_bhdr> Data 0... }
Record #N+1 { <raw_prod_bhdr> <ingest_data_header(s)> Data... }
Record #N+2 { <raw_prod_bhdr> Data... }
. . .
. . .
Record #M { <raw_prod_bhdr> Data 0... }
```

4.6.6.1 Data compression algorithm

To make the best use of storage, radar rays are compressed before being inserted into the file.

The compression algorithm is 16-bit word based, and removes runs of zeros.

This complements the signal processor, which zeros data that does not meet the threshold requirements in effect.

Runs of one or two zeros are not removed because there is no benefit. The data field starts with a compression code value.

The code either indicates the number of zeros that were skipped, or the number of data words that follow. In the case of a zero skipped code, it is immediately followed with another code value. In the case of a data code, the next code follows the data.

Table 15 Compression code meanings

MSB	Low-bits	Meaning
0	0	<unused>
0	1	End of ray
0	2	<unused>
0	3 ... 32767	3 ... 32767 zeros skipped
1	0	<unused>
1	1 ... 32767	1 ... 32767 data words follow

4.6.6.2 Raw Product Example

This is an example of a third record of a simple raw product.

This product is for data with only velocity recorded. Shown is the first ray of a PPI at azimuth 0 ... 1°, elevation 0.5°. It has 200 range bins, with no data for the first 100 bins, then one bin of zero velocity, then 99 bins with no data.

Table 16 Raw product example

Byte	Size	Contents	
0	12	<raw_prod_bhdr>	
12	76	<ingest_data_header>	
88	SINT2	-32762 (code for six data words to follow)	
90	BIN2	0 (starting azimuth)	Six words referred to above
92	BIN2	91 (starting elevation)	
94	BIN2	182 (ending azimuth)	
96	BIN2	91 (ending elevation)	
98	SINT2	200 (number of range bins)	
100	UINT2	3 (time)	
102	SINT2	50 (code for fifty zeros skipped)	
104	SINT2	-32767 (code for one data word follows)	
106	SINT2	128 (zero velocity value)	One data word
108	SINT2	49 (code for forty nine zeros skipped)	

Byte	Size	Contents	
110	SINT2	1 (code for end of ray)	
112			
		(Continues on to next ray)	

4.6.7 SLINE product format

The first 1024 bytes of the product hold a copy of the **protect_setup** structure described in protect_setup structure (page 45).

This is followed by an array of **sline_results** structures one for each shearline found.

The **Number of elements in product results array** element of the **product_end** structure in the **product_hdr** indicates the number of points in the array. Generally there is at most 1 shearline found.

4.6.8 TDWR product format

The first 1024 bytes of the product hold a copy of the **protect_setup** structure described in protect_setup structure (page 45).

This structure is copied from the setup files on the integrating computer.

This is followed by:

1. An array of **tdwr_results** structures, one for each corridor.
   The **Number of elements in product results array** element of the product_end structure in the **product_hdr** indicates the number of elements in the array. Only corridors covered by one of the input products to the integrator are included, and corridors which are unused at generation time are normally be removed.
2. An array of **warning_results** structures copied from the WARN input product, if any.
3. An array of **sline_results** structures copied from the SLINE input product, if any.
   The sizes of these arrays are in the **tdwr_psi_struct** portion of the product header.

4.6.9 TRACK product format

The first 1024 bytes of the product hold a copy of the protect_setup structure described in protect_setup structure (page 45).

This is followed by an array of track_results structures, one for each track point.

The **Number of elements in product results array** element of the **product_end** structure in the **product_hdr** indicates the number of points in the array.

Points must be in time order, with the oldest first. Within points of the same time, they are sorted by index number. Only one data point of each index value at each time is allowed, except that multiple text points are allowed (which have index set to 0).

4.6.10 VAD product format

The velocities produced by the **VAD** product are stored in an array of `vad_results` structures.

This is a 2D array of results: First included are all the results from the first elevation sweep, followed by the second, and so on.

You can use the structure `vad_product` defined in `product.h` to reference this data.

Only data bins with valid velocities are included in the average. For hybrid inputs, it is possible that the number of azimuths are different for different elevations.

4.6.11 VVP product format

The winds produced by the VVP product are stored in an array of `vvp_results` structures for each height.

If less than 30 range bins are found in the height interval, then the whole structure is zeroed. Therefore any analysis program should check the `Number of data points used` field. Also if the calculation cannot be performed for some other reason, the standard deviation is set to 32767.

Analysis programs should also check the standard deviations. Data fields which are turned off in the product configuration are set to 0. Therefore the `Wind parameters mask` in the `vvp_psi_struct` in the product header must be checked. Only data bins with valid velocities not near 0 are included in the calculation.

To compute the average reflectivity, bins must also have a valid reflectivity. Because of this, it can produce a lower number of valid bins, so the number of valid reflectivity bins is also recorded. Same for RhoHV.

4.6.12 WARN product format

The first 1024 bytes of the product hold a copy of the `protect_setup` structure described in protect_setup structure (page 45).

This is followed by an array of `warning_results` structures.

The number of `warning_results` structures in the array is given in the `product_end` structure, part of the `product_hdr`.

4.6.13 WIND product format

The first 84 bytes of the product hold a `vvp_results` structure for the whole volume.

This is followed by an array of `wind_results` structures. The number of `wind_results` structures in the array is determined by multiplying the number of points in range by the number of points in azimuth stored in the `product_specific_info` structure in the header.

4.7 Tape format

Tapes made by IRIS hold exact images of corresponding disk-based product files. The tapes are always written using fixed-length 6144-byte records, where the last tape record is padded with zeros, if necessary, to the full 6144-byte length.

Products are separated on tape by end-of-file (EOF) marks.

Because disk product files begin with a product_hdr structure, this is also the structure initially encountered in the first record of each product on tape. By examining the headers, you can determine what kinds of product files have been stored on the tape.

There is a special short record at the beginning of the tape that serves to identify how and when the tape was initially created by the `init_iris_tape` utility. This record contains the tape_header_record structure, and is followed by an EOF and the product files, if any. There are no special directory or inventory records on the tape.

After a tape has been started, the only additional writing that can be done is to append more product file images to the end.

4.8 TIFF output format

IRIS can output images over the network in TIFF format.

These files conform to the TIFF revision 6.0 standard. While the standard supports many image types, IRIS uses only baseline TIFF, and none of the TIFF extensions. Only one image is included in each file, the Palette Color image.

The following table lists the fields set by IRIS. Compression is controlled by a setup question for the output device.

Table 17 TIFF fields used by IRIS

Field	Code	Description
`Software`	305	Example: `IRIS 5.56`
`ImageDescription`	270	Example: `IRIS PPPPP NNNNNNNNNNNN` Contains the 5 character product type, followed by the 12 character product configuration name.
`DateTime`	306	Time of ingest data
`ImageWidth`	256	Width of image in pixels
`ImageLength`	247	Height of image in pixels
`Compression`	259	Either none or PackBits
`PlanarConfiguration`	284	1 (Chunky)
`SamplesPerPixel`	277	1
`Orientation`	274	1 (Top Left)
`RowsPerStrip`	278	Height of image in pixels

4.9 Constants

Extended Header (**DB_XHDR**) is included here though it is not generated by the DSP. In general, types 0 ... 31 can be produced by the DSP.

Table 18 Data type constants — */include/sigtypes.h*

Data type	Constant	Description
DB_XHDR	(0)	Extended Headers
DB_DBT	(1)	Total power (1 byte)
DB_DBZ	(2)	Reflectivity (1 byte)
DB_VEL	(3)	Velocity (1 byte)
DB_WIDTH	(4)	Width (1 byte)
DB_ZDR	(5)	Differential reflectivity (1 byte)
DB_DBZC	(7)	Corrected reflectivity (1 byte)
DB_DBT2	(8)	Total power (2 byte)
DB_DBZ2	(9)	Reflectivity (2 byte)
DB_VEL2	(10)	Velocity (2 byte)
DB_WIDTH2	(11)	Width (2 byte)
DB_ZDR2	(12)	Differential reflectivity (2 byte)
DB_RAINRATE2	(13)	Rainfall rate (2 byte)
DB_KDP	(14)	**KDP** (Differential phase) (1 byte)
DB_KDP2	(15)	**KDP** (Differential phase) (2 byte)
DB_PHIDP	(16)	**PhiDP** (Differential phase) (1 byte)
DB_VELC	(17)	Corrected velocity (1 byte)
DB_SQI	(18)	**SQI** (1 byte)
DB_RHOHV	(19)	**RhoHV** (1 byte)
DB_RHOHV2	(20)	**RhoHV** (2 byte)
DB_DBZC2	(21)	Corrected Reflectivity (2 byte)
DB_VELC2	(22)	Corrected velocity (2 byte)
DB_SQI2	(23)	**SQI** (2 byte)
DB_PHIDP2	(24)	**PhiDP** (Differential phase) (2 byte)
DB_LDRH	(25)	**LDR** `xmt H, rcv V` (1 byte)
DB_LDRH2	(26)	**LDR** `xmt H, rcv V` (2 byte)

Data type	Constant	Description
DB_LDRV	(27)	LDR xmt V, rcv H (1 byte)
DB_LDRV2	(28)	LDR xmt V, rcv H (2 byte)
. . .		
DB_HEIGHT	(32)	Height (1/10 km) (1 byte)
DB_VIL2	(33)	Linear liquid (.001mm) (2 byte)
DB_RAW	(34)	Raw Data
DB_SHEAR	(35)	Wind Shear (1 byte)
DB_DIVERGE2	(36)	Divergence (2 byte)
DB_FLIQUID2	(37)	Floated liquid (2 byte)
DB_USER	(38)	User type, unspecified data (1 byte)
DB_OTHER	(39)	Unspecified data, no color legend (1 byte)
DB_DEFORM2	(40)	Deformation (2 byte)
DB_VVEL2	(41)	Vertical velocity (2 byte)
DB_HVEL2	(42)	Horizontal velocity (2 byte)
DB_HDIR2	(43)	Horizontal wind direction(2 byte)
DB_AXDIL2	(44)	Axis of dilatation (2 byte)
DB_TIME2	(45)	Time in seconds (2 byte)
DB_RHOH	(46)	Rho, xmt H, rcv V (1 byte)
DB_RHOH2	(47)	Rho, xmt H, rcv V (2 byte)
DB_RHOV	(48)	Rho, xmt V, rcv H (1 byte)
DB_RHOV2	(49)	Rho, xmt V, rcv H (2 byte)
DB_PHIH	(50)	Phi, xmt H, rcv V (1 byte)
DB_PHIH2	(51)	Phi, xmt H, rcv V (2 byte)
DB_PHIV	(52)	Phi, xmt V, rcv H (1 byte)
DB_PHIV2	(53)	Phi, xmt V, rcv H (2 byte)
DB_USER2	(54)	User type, unspecified data (2 byte)
DB_HCLASS	(55)	Hydrometeor class (1 byte)
DB_HCLASS2	(56)	Hydrometeor class (2 byte)
DB_ZDRC	(57)	Corrected differential reflectivity (1 byte)
DB_ZDRC2	(58)	Corrected differential reflectivity (2 byte)
. . .		
DB_PMI8	(75)	Polarimetric meteo index (1 byte)
DB_PMI16	(76)	Polarimetric meteo index (2 byte)

Data type	Constant	Description
DB_LOG8	(77)	Log receiver signal-to-noise ratio (1 byte)
DB_LOG16	(78)	Log receiver signal-to-noise ratio (2 byte)
DB_CSP8	(79)	Doppler channel clutter power (1 byte)
DB_CSP16	(80)	Doppler channel clutter power (2 byte)
DB_CCOR8	(81)	Cross correlation, uncorrected rhohv (1 byte)
DB_CCOR16	(82)	Cross correlation, uncorrected rhohv (2 byte)
DB_AH8	(83)	Attenuation of Zh (1 byte)
DB_AH16	(84)	Attenuation of Zh (2 byte)
DB_AV8	(85)	Attenuation of Zv (1 byte)
DB_AV16	(86)	Attenuation of Zv (2 byte)
DB_AZDR8	(87)	Attenuation of Zzdr (1 byte)
DB_AZDR16	(88)	Attenuation of Zzdr (2 byte)

5. Information utilities

5.1 Productx

The **productx** product examiner utility displays the information contained in a specified product file.

For all product types, **productx** displays:

1. Product header, including information such as the site where the ingest data came from, the date and time when the ingest data was gathered, and its size.
2. Product-specific meta-data from the header for many product types.
3. Data values from the file.
   For Cartesian data files, this could be a large number of pixels. In this case it skips data to present a summary display which fits on the terminal. If you want no skipping, specify a very large terminal width, say 10000.

5.1.1 Invoking productx

To invoke **productx**, issue the command: **`productx [options] filename`**

`filename` is the name of a product file stored in */usr/iris_data/product*.

Raw products are stored in a separate directory: */usr/iris_data/product_raw*.

Table 19 Raw Product Parameters

Parameter	Description
`-help`	Prints the list of options.
`<filename>`	Specify which ingest header file to read.
`-sweep:#`	Specify sweep number, origin 1. Z-axis for 3D data.
`-version`	Print out just the version number.
`-width=#`	Specify maximum line with for data display.

Each product file has a unique name based on the site ID, date, and a randomized algorithm. The first 3 letters of the file extension show the product type.

Table 20 Product file naming

File name label	Product type
BAS	**BASE**
BEA	**BEAM**
CAP	**CAPPI**
FCA	**FCAST**

File name label	Product type
CAT	**CATCH**
DWE	**DWELL**
HMX	**HMAX**
IMG	**IMAGE**
LAY	**LAYER**
MAX	**MAX**
MHG	**MELTING LAYER**
NDP	**NDOP**
OTH	**OTHER**
PPI	**PPI**
RAW	**RAW**
RN1	**RAIN1**
RNN	**RAINN**
RHI	**RHI**
RTI	**RTI**
SHE	**SHEAR**
SLI	**SLINE**
STA	**STAT**
TDW	**TDWR**
THK	**THICK**
TOP	**TOPS**
TRA	**TRACK**
TXT	**TEXT**
USE	**USER**
VAD	**VAD**
VUS	**VUSER**
VIL	**VIL**
VVP	**VVP**
WND	**WIND**
WRN	**WARN**
XSE	**XSECT**

5.1.2 Productx examples

PPI example

A **PPI** product shows useful header information:

```
$ productx TMS090520180345.PPIFD6W -width:80
------------- Product Summary for TMS090520180345.PPIFD6W -------------
Ingest site name :'tms_rad1', Version: 8.11
Ingest hardware name :'tms_rad1'
Product site name :'SIGMET, dry2', Version: 8.12
File size: 519040 bytes (Disk space: 519040 bytes)
Product type is: PPI
PCO name: DEF_DBZ, TCO name: PPIVOL_B
PRF: 840/560Hz, Wavelength: 10.63cm, Nyquist: 44.65m/s(V), 22.32m/s(W)
XMT Polarization: Horizontal, Wind:???
Constant:72.06 dB, I0:-104.82 dBm, Cal Noise:-70.66 dBm, Bandwidth:0 kHz.
ZFlags: SP_T, block_zc, attn_zc, target_zc, dpatten_zc, dpatten_z
VFlags: SP_V, 3lag_w, ship_v, unfold_vc, FALL_VC, storm_vc
Heights: Radar: 970m, Ground: 948m, Melting: 4900m MSL
Size is: 720x720x1 pixels
Scale is: 500.00 x 500.00 x 0.00 m/pixel
Center Location: 22 24.7'N, 114 7.4'E, ref: 0 meters
Projection type is: Eqdist Cylinder
Projection Reference Point: 22 24.7'N, 114 7.4'E
Equitorial Radius: 0.00000 km, Flattening: 1/0.00000
Radar position is: 360.0, 360.0 pixels
Product data type is dBZ (2)
Color count:16, Color set: 1, variable
Seams: 16452.00 16702.00 16952.00 17252.00 17402.00 17502.00 17752.00
18052.00
18202.00 18302.00 18552.00 18852.00 19002.00 19102.00 19352.00 19652.00
Maximum range: 180.0 km
PPI elevation angle: 6.60 degrees
Ingest time: 18:03:45 20 MAY 2009 HKT (-480 minutes west) DST:0/0
Volume scan time: 18:03:45 20 MAY 2009 HKT (LT: HKT -480 minutes)
Oldest Ing time: 18:03:45 20 MAY 2009 LT
Product Gen time: 19:23:31 17 JUN 2009 UTC
Input count: 1
Product is not composited.
Displaying cartesian data with skip factor 40
```

```
719:255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
679:255 255 255 255 255 255 0   0   0   0   0   0   0   255 255 255 255 255
639:255 255 255 255 0   0   0   0   0   0   0   0   0   0   0   255 255 255
599:255 255 255 0   0   0   0   0   0   0   0   0   0   0   0   0   255 255
559:255 255 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   255
519:255 255 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   255
479:255 0   0   0   0   0   0   0   105 117 104 66  0   0   0   0   0   0
439:255 0   0   0   0   0   82  85  121 57  0   101 81  0   0   0   0   0
399:255 0   0   0   0   0   83  111 136 78  0   0   0   0   0   0   0   0
359:255 0   0   0   0   0   83  169 71  0   0   0   0   0   0   0   0   0
319:255 0   0   0   0   0   133 87  0   0   0   0   0   0   0   0   0   0
279:255 0   0   0   0   0   97  93  0 0 0   0   0   0   0   0   0   0
239:255 0   0   0   0   0   78  77  64  0   0   0   0   0   0   0   0   0
199:255 255 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   255
159:255 255 0   0   0   0   0   0   0   0   0   255 0   0   0   0   0   255
119:255 255 255 0 0 0   0   0   0   0   0   0   0   0   0   0   255 255
79: 255 255 255 255 0   0   0   0   0   0   0   0   255 0   0   255 255 255
39: 255 255 255 255 255 255 0   0   0   0   0   0   0   255 255 255 255 255
```

`Status` example

A `Status` product shows the status of IRIS processes at a particular site.

```
$ productx HOT000602152122.STAZ0GN
------------- Product Summary for HOT000602152122.STAZ0GN -------------
Ingest site name : 'SIGMET, HOT', Version: 7.17
Ingest hardware name : 'SIGMET, HOT'
Product site name : 'SIGMET, HOT', Version: 7.17
File size: 2340 bytes (Disk space: 2340 bytes)
Product type is: Status
PCO name: SIGMET, HOT, TCO name: FAULT
PRF: 500Hz, Wavelength: 5.00cm, Nyquist: 6.25m/s(V), 6.25m/s(W)
Polarization: Horizontal, wind:???
Heights: Radar: 600m, Ground: 100m, Melting: ???m MSL
Size is: 0x0x0 pixels
Center Location: 42_33.0'N, 71_25.8'W, ref: 600 meters
Projection type is: Azimuthal Equidistant
Projection Reference Point: 42_33.0'N, 71_25.8'W
Radar position is: 0.0, 0.0 pixels
Scale is: 0.000 x 0.000 x 0.000 km/pixel
Product data type is Xhdr (0)
Maximum range: 0.0 km
Ingest time: 15:21:22 2 JUN 2000 UTC (0 minutes west) DST:0/1
Volume scan time: 15:21:22 2 JUN 2000 (LT: EDT 300 minutes)
Oldest Ing time: 15:21:22 2 JUN 2000
Product Gen time: 15:21:22 2 JUN 2000
Input count: 1
Product is not composited
Site style is: RADAR
```

```
Overall status is: FAULT
Status of IRIS_INGEST ON/Idle
Status of IRIS_INGFIO ON/NA
Status of IRIS_OUTPUT ON/NA
Status of IRIS_PRODUCT ON/Idle
Status of IRIS_WATCHDOG ON/Running
Status of IRIS_REINGEST ON/Idle
Status of IRIS_NETWORK ON/Idle
Status of IRIS_NORDRAD OFF/Stopped
Status of IRIS_SERVER ON/Idle
Status of IRIS_RIBBON OFF/Stopped
Status of IRIS_INPUT ON/Idle
Status of DSP N/A Stopped
Status of RCP Critical N/A DEAD
Status of WINDOW1 OK Idle test
Status of NETWORK1 OK Idle ToArchive2
Status of NETWORK2 OK Idle netcdf_out
Status of NETWORK3 OK Idle ToBufr
Status of NETWORK4 OK Idle ToHDF5
Status of WINDOW2 OK Idle Joe
Status of ARCHIVE1 OK Idle archive
RST mode: 'DEFAULT'
TSC mode: 'DEFAULT'
PSC mode: 'DEFAULT'
POM mode: 'DEFAULT'
```

```
Active task: ''
Active product: ''
Antenna Position, azimuth: 20.00, elevation: -0.99
Bite fault summary shows 2
Low Airflow: OK
Interlock: OK
Waveguide: OK
Top message #9, Repeats: 1
Problem starting scan at EL=6 (AZ velocity out of range)
Process: IRIS_INGEST, Name: F:202 M:3
Time: 16:34:36 30 MAY 2000
Message list contains 0 messages:
```

5.2 Rays utility

The **rays** utility gives information about ingest files. You can choose to display information about various parameters of the ingest data.

5.2.1 Invoking rays

To invoke **rays**, issue the command:

```
rays [options] filename
```

where *filename* is the name of an ingest header file stored in the directory */usr/iris_data/ingest*.

Ingest files are named with a time stamp for when the data were gathered. For example, ingest data gathered at 10:17:30 on December 2, 1994 is stored in a file named *941202101730*.

Table 21 Ray options

Option	Description
`-help`	Prints the list of options.
`-data:dtype`	Specify which data type to display.
`[-if:]filename`	Specify which ingest header file to read.
`-inter`	Run in interactive mode. This is the old style, **rays** then prompts for some of the options.
`-perf`	Performance test, display ray headers only.
`-range:#`	Specify starting range in km.
`-sweep:#`	Specify sweep number, origin 1.
`-terse`	Skip showing ingest header info.
`-width:#`	Specify maximum line with for data display.

To make it easier to enter the names of the ingest files, change your default directory to the ingest directory and get a listing of all the header files (with `ls *`.). Select the name of a file with the mouse and paste it into the command line.

5.2.2 Headers only example

In the following example, **rays** uses the **-perf** option to display the ray header information only. This format lets you evaluate the speed of IRIS or the signal processor and look for missing rays.

There are 2 azimuth and elevation angles recorded with each ray. These are angles at the beginning and ending of each ray.

```
$ rays -perf -terse DRY020418160516. | more
Reading file: /usr/iris_data/ingest/DRY020418160516.01dBT

# 0 Az: 359.80,0.99 El: 0.48, 0.48 Size: 967 16:05:33
# 1 Missing
# 2 Az: 0.99, 2.20 El: 0.48, 0.48 Size: 967 16:05:33
# 3 Az: 2.20, 3.38 El: 0.48, 0.48 Size: 967 16:05:33
# 4 Az: 3.38, 4.59 El: 0.48, 0.48 Size: 967 16:05:33
# 5 Az: 4.59, 5.78 El: 0.48, 0.48 Size: 967 16:05:34
# 6 Az: 5.78, 6.99 El: 0.48, 0.48 Size: 967 16:05:34
# 7 Missing
# 8 Az: 6.99, 8.17 El: 0.48, 0.48 Size: 967 16:05:34
# 9 Az: 9.38, 9.38 El: 0.48, 0.48 Size: 967 16:05:34
# 10 Az: 9.38, 10.57 El: 0.48, 0.48 Size: 967 16:05:34
# 11 Az: 10.57, 11.78 El: 0.48, 0.48 Size: 967 16:05:34
# 12 Az: 11.78, 12.96 El: 0.48, 0.48 Size: 967 16:05:34
# 13 Missing
# 14 Az: 14.17, 14.17 El: 0.48, 0.48 Size: 967 16:05:34
# 15 Az: 14.17, 15.36 El: 0.48, 0.48 Size: 967 16:05:34
# 16 Az: 15.36, 16.57 El: 0.48, 0.48 Size: 967 16:05:34
# 17 Az: 16.57, 17.75 El: 0.48, 0.48 Size: 967 16:05:34
# 18 Az: 17.75, 18.96 El: 0.48, 0.48 Size: 967 16:05:34
# 19 Az: 18.96, 18.96 El: 0.48, 0.48 Size: 967 16:05:34
# 20 Az: 18.96, 20.15 El: 0.48, 0.48 Size: 967 16:05:34
# 21 Az: 20.15, 21.36 El: 0.48, 0.48 Size: 967 16:05:34
# 22 Az: 21.36, 22.54 El: 0.48, 0.48 Size: 967 16:05:34
# 23 Az: 22.54, 23.75 El: 0.48, 0.48 Size: 967 16:05:34
# 24 Az: 23.75, 24.94 El: 0.48, 0.48 Size: 967 16:05:34
```

5.2.3 Velocity example

You can choose any of the data parameters — **V** in this example — and rays then displays the header, followed by range bins starting from the specified bin.

Where no data is available for a ray, it displays **Missing**. When no data is available for a range bin within the ray, it displays a series of dashes (`--.- --`).

```
$ rays XXX041029121855. -data:v2 | more
[joe@localhost ingest]$ rays COX071115233217. -data:v | more
Task Summary for: COX071115233217.
Site name: 'cox-radar', Task name: 'PPI_C'
Scan: PPI, Speed: 9.00 deg/sec, Resolution:1.00 deg
Description: 'Doppler Velocity Volume Scan'
Location: 21 26.0'N 91 58.6'E, Altitude: 64 meters, Melting height:Unknown
Dpolapp config:
Volume Time: 23:32:17.442 15 NOV 2007 UTC (0 min. west) (LT: BDT -360 min.)
ZFlags: SP_T, block_zc, attn_zc, target_zc, dpatten_zc, dpatten_z
VFlags: SP_V, 3lag_w, ship_v, unfold_vc, fall_vc, storm_vc
PRF: 720/576Hz, PulseWidth: 1.00 usec (1)
BeamWidth: 1.46/1.50 deg.
Radar constant: 0.00/0.00 dB, Receiver bandwidth 0 kHz.
Calibration I0: -111.95/-113.03 dBm, with noise -79.70/-76.95 dBm.
LOG-Noise: 0.1776, Lin-Noise: 0.1776, I-Off: 0.0000, Q-Off: 0.0000
SOPRM Flags: 0x04af, LOG Slope: 0.480, Z-Cal: -36.81dBZ, H/V: 0.00 dB
Filters: Dop:6, Log:0; PntClt: 3, Thresh: 1.0 dB; Samples: 80
Processing Mode: FFT, Xmt Phase: Fixed
Zdr Threshold: LOG GDR = 0.00 dB, XDR = 0.00 dB
T Threshold: LOG LOG = 2.4 dB
Z Threshold: LOG & CSR SIG = 5.0 dB
V Threshold: SQI & CSR CSR = 15.0 dB
W Threshold: SIG & SQI & LOG SQI = 0.42
Available moments are: dBZ V
Original moments were: dBT dBZ V W
Starting range 0.125 km, range bin spacing 625 meters
There are 8 sweeps, each having 360 rays and 320 bins
Angle list: 0.0 1.5 3.0 4.5 6.0 9.0 12.0 15.0
Reading file: /usr/iris_data/ingest/COX071115233217.01V
Sweep Time: 23:32:17.442 15 NOV 2007
Starting at range 0.12 km (bin 1), bin step: 0.62 km
# 0 Az: 359.52, 0.47 El: 0.02, 0.02 Size: 320 23:32:32
--.- --.- --.- --.- --.- --.- --.- 17.3 17.3 17.3 17.3 --.- --.- --.-
# 1 Az: 0.47, 1.51 El: 0.02, 0.02 Size: 320 23:32:32
--.- --.- --.- -61.5 -61.5 -62.1 -62.1 -62.7 59.7 59.1 --.- --.- --.- --.-
# 2 Az: 1.53, 2.52 El: 0.02, 0.02 Size: 320 23:32:32
--.- --.- --.- --.- --.- --.- --.- --.- --.- --.- --.- --.- --.- --.-
# 3 Az: 2.54, 3.50 El: 0.02, 0.02 Size: 320 23:32:32
--.- --.- --.- --.- --.- --.- --.- --.- --.- 34.6 -38.8 -1.8 --.- --.-
# 4 Az: 3.52, 4.53 El: 0.02, 0.02 Size: 320 23:32:33
--.- --.- -31.0 -31.0 -31.0 -31.0 -31.0 -30.4 -30.4 -30.4 56.1 -1.8 -1.8 -
1.8
--More--
```

5.2.4 Extended header example

For shipboard systems, additional information for each ray can be stored in an extended header in the ingest file.

To display the extended header, use the **`-data:xhdr`** option.

```
$ rays -data:xhdr -terse -if:DRY020418160516. | more
Reading file: /usr/iris_data/ingest/DRY020418041055.01Xhdr
```

The extended header includes time recorded to the nearest millisecond.

```
Starting at range 0.00 km (bin 1), bin step: 0.30 km
# 0 Az: 359.80, 0.55 El: 0.42, 0.42 Size: 1 4:11:08.055
```

The extended header shows the antenna azimuth and elevation, the platform pitch, roll, and heading, and the derivatives of these values in degrees/second:

```
Az: 350.95 El: 0.40 Pitch: -0.13 Roll: -1.10 Head: 346.79
Vel: 18.72 deg/s 0.04 0.70 359.82 359.82
```

The extended header information is recorded from a serial data stream transmitted from RCP. Typically this is configured to transmit updates at a maximum speed of about 20 times per second. Because rays can be recorded at up to 40 rays per second, and because of pipeline delays in the serial data, the extended header azimuth can lag the actual azimuth by up to several degrees. Normally, the platform motion period is slower.

```
Tr: 2.29 El_or: 0.31 Lat: 1#section#45.5'N Long:138#section#
2.8'E Alt: 14
Cor: 3.59 Age: 291 Vel: 3.69 m/s 0.00 -0.17
```

`Tr`
: Training angle, which is the pedestal relative azimuth of the pedestal.

`El_or`
: Pedestal relative elevation angle.

`Cor`
: Velocity correction (meters/second). Applied to velocity data to correct for platform motion.

`Age`
: Time in milliseconds since this update arrived from RCP.

`Vel`
: Platform position and motion. Altitude is in meters, and motions are in meters/second.

More information

- extended_header_v1 structure (page 29)

Appendix A. structmap command

Use the **structmap** command to display the format of IRIS structures. This is useful when writing applications that access IRIS data.



To make **structmap** available on your system, install IRIS with the **-headers** option .

structmap options

To show a list of **structmap** options, enter the command without options or parameters:

```
$ structmap
Command Line Options:
<struct name> :   Display internal contents of IRIS structure(s)
-include <dir> :  Override default 'include' directory name
-nopack :         Force no packing of structure elements
-scan :           Produce list of all defined structures
-scanlocal :      Like 'scan', but do local directory only
-noflags :        Suppress error flags in output
-recursive :      Descend into substructures
-data :           Show numeric data read from std.input
-dimension N :    Use with '-data' for N-dimensional printout
```

For example **structmap 'structmap -scan'** displays everything.

structmap <structure name> option

Invoke **structmap** with the name of a structure to display the name of the include file where the structure is defined and a description of each element in the structure.

It offsets from the beginning of the structure, its size, the number of times it occurs, its data type, and name.

For example, to display information about the **tape_header_record** structure:

```
$ structmap tape_header_record

tape_header_record /usr/include/irisrda/output.h
0       12        1    struct structure_header hdr
12      16       16    char stape_id[]
28      16       16    char sitename[]
44      12        1    struct ymds_time init_time
56       2        1    SINT2 idrive
58       2        2    char ipad58x2[]
60       8        8    char sversion[]
68     252      252    char ipad_end[]
320
```

The example shows that the structure is defined in */include/irisrda/output.h* and contains:

- **hdr**, a structure of type **structure_header**, taking up the first 12 bytes.
- **stape_id** and **sitename**, arrays of 16 characters each, at offsets 12 and 28.
- **init_time** is a **ymds_time**, structures taking up 12 bytes starting at offset 44.
- **idrive**, a long integer at offset 56.
- **ipad58x2**, **sversion**, and **ipad_end**, arrays of 2, 8, and 252 characters, at offsets 58, 60, and 68, respectively.

The total size of the structure is 320 bytes.

–scan Option

The **–scan** option lists the names of the structure defined by IRIS.

You can use the **–scan** option to recursively call **structmap** and display the format of all the structures in the system.

```
$ structmap -scan
ant_manual_setup
bitex_field_def
bitex_top_def
cappi_psi_struct
.
.
.
```

To redirect the output to a file, type:

```
$ structmap 'structmap -scan' > allstructs.out
```

Appendix B. Radar Control Protocol

The interface to the Radar Control Protocol (RCP) is either over:

- Network UDP multi-cast packet
- Serial line
- UNIX FIFO

When using the network, transmissions to both directions are sent to the same address and port number. This allows the seamless merging of data from multiple clients. It also means that there is not a separate wire for transmit and receive as with a serial cable. The data format consists of the serial format with a 16-byte prefix.

The prefix consists of the packet size in ASCII (8-bytes), followed by a ASCII code for the type of packet. The first letter of the type is either `T` (transmit) or `R` (receive). This is relative to the main controlling host. When coding, you must filter on the direction.

For examples, see the *`base/ant_utils/antx.C`* program, as well as the *`base/antenna_lib/ant_netrcv.c`* and *`base/user_lib/UdpSupport.c`* files.

Specifying the interface

To code multicast packets over broadcast packets, you must specify which interface to use, and issue the `IP_ADD_MEMBERSHIP` call. Vaisala recommends using `site-local` addresses.

When using a 2-way asynchronous RS-232 data line, the baud rate is typically 9600 (19200+ for shipboard systems). IRIS uses this link to control the servo and antenna and receive feedback status.

Information is transferred in packets of 2 or more bytes. Each packet begins with a `SYNC` byte and ends with an `END` byte of `FF(Hex)`. All `SYNC` bytes have the MSB set, and the particular value indicates the type of packet to follow. Types include, for example, `80(hex)` for antenna, `C0(hex)` for BITE, `AF(hex)` for Q-BITE, and `B0(hex)` for time.

Each of packet type has a specific direction of travel (to or from IRIS), but packets can arrive in any order within the serial stream.

Antenna communication formats

Older systems use the RCV01 and XMT01 formats. Newer systems can use the RCV02 and the XMT02 formats.

The RCV03 format is intended for systems on moving platforms, such as ships or airplanes. To correct the radar's measured radial velocity for the motion of the platform, the 3-D velocity and orientation of the platform must be recorded. Typically, the information comes from an inertial navigation system. For shipboard system, an update rate of approximately 20 reports per second can satisfy the velocity correction requirements at 19200 baud.

Angles

The latitude and longitude are transmitted as 21-bit binary angles. The following angles are transmitted as 14-bit binary angles:

- azimuth and elevation
- train order
- pitch, roll, and heading

In the XMT01 format, the angular speed is a signed number in units of 0.55º/sec. In all other formats, the angular rates are in signed 14-bit binary angles per second. The largest possible value is 180º/sec (30 rpm) and the step is 0.022º/sec. All velocities are in signed cm/sec with the altitude in signed meters. If some of the information is unavailable at the full resolution of the data format, the low bits are filled with zeros.

The azimuth and the elevation angles are corrected angles relative to the north and are the angles that the antenna is pointed relative to the deck of the platform. These calculations are derivable from the other angles but are also reported to assist in the data analysis, especially if one of the sensors or the stabilization fails.

The pitch is the angle between the fore-and-aft axis of the platform and the horizontal is measured in the vertical plane. The pitch is positive when the bow is down and the roll is the rotation angle about the fore-and-aft axis in its pitched position. The pitch is measured in the plane perpendicular to the fore-and-aft axis, which is generally not the vertical plane, and the roll is positive when the deck is down on the port side.



Note: The pitch can be directly measured by a level on the fore-and-aft axis but the roll cannot be directly measured by a one-axis tilt meter.

The heading is referred to as the direction the platform is pointed but is not the same as direction of motion. The platform could be pointed one way and drifting backwards.

The time stamp is a 14-bit counter incremented by the RCP once per millisecond. The RCP must latch all the data for a packet at the same time. This counter allows the host computer to accurately judge the time between samples without the serial line latencies and fluctuations due to the time sharing operating system.

The position of the platform is reported by the latitude, the longitude, and the altitude. Since the altitude may not be implemented for systems on ships, the setting is 0.

Status and command packets

Table 22 Status packet RCV01 format (RCP to host)

Char	Function
1	SYNC Byte (80 Hex)
2	Azimuth Low 7 bits
3	Azimuth High 7 bits
4	Elevation Low 7 bits

Char	Function
5	Elevation High 7 bits
6	Status #1 D6 = Low air flow D5 = Low Waveguide pressure D4 = Servo power D3 = Antenna Local mode D2 = Interlock D1 = Standby D0 = Radiate On
7	Status #2 D6 = RCP02 is shutdown D5 = LSB pulse width D4 = T/R power On D3 = T/R Local mode D2 = Encoders calibrated D1 = MSB pulse width D0 = Magnetron current normal
8	End Of Message (FF Hex)

Table 23 Control packet XMT01 format (host to RCP)

Char	Function
1	SYNC Byte (80 Hex)
2	Azimuth Low 7 bits
3	Azimuth High 7 bits
4	Elevation Low 7 bits
5	Elevation High 7 bits
6	Control Word #1 D6 = MSB of Pulse Width D5 = Leave Pulse width unchanged D4 = Spare D3 = Signal Generator On D2 = Signal Generator CW D1 = EL (1 = Scan, 0 = Position) D0 = AZ (1 = Scan, 0 = Position)

Char	Function
7	Control Word #2 D6 = Reset RCP02 on edge D5 = Noise Source On D4 = LSB of Pulse width D3 = Radiate On complemented D2 = Radiate On D1 = Servo Power On D0 = T/R Power On
8	Control Word #3 (all spare)
9	Signal generator level (unsigned 0 ...127dB attenuation)
10	AZ/EL Antenna speed (signed 7 bit, 0.55° resolution)
11	End Of Message (FF Hex)

Table 24 Status packet RCV02 / RCV04 format (RCP to host)

Char	Function
1	SYNC Byte (80 Hex)
2	Azimuth Low 7 bits
3	Azimuth High 7 bits
4	Elevation Low 7 bits
5	Elevation High 7 bits
6	Azimuth Rate Low 7 bits
7	Azimuth Rate High 7 bits
8	Elevation Rate Low 7 bits
9	Elevation Rate High 7 bits
10	Status #1 D6 = Low air flow D5 = Low Waveguide pressure D4 = Servo Power D3 = Antenna Local mode D2 = Interlock Open D1 = Standby D0 = Radiate On

Char	Function
11	Status #2 D6 = RCP02 is shutdown D5 = LSB pulse width D4 = T/R Power On D3 = T/R Local mode D2 = Azimuth encoder calibrated D1 = MSB pulse width D0 = Magnetron current normal
12	Status #3 D6 = IRIS Mode 2 D5 = IRIS Mode 1 D4 = IRIS Mode 0 D3 = Elevation encoder calibrated D2 = Signal Generator fault D1 = Signal Generator On D0 = Signal Generator CW
13	Signal generator level (0=max power)
14	Time Stamp Low 7 bits
15	Time Stamp High 7 bits
16	End Of Message (FF Hex)

Table 25 Control packet XMT02 / XMT04 format (host to RCP)

Char	Function
1	SYNC Byte (80 Hex)
2	Azimuth Low 7 bits
3	Azimuth High 7 bits
4	Elevation Low 7 bits
5	Elevation High 7 bits

Char	Function
6	Control Word #1 D6 = MSB of Pulse Width D5 = Leave Pulse Width unchanged D4 = Spare D3 = Signal Generator On D2 = Signal Generator CW D1 = EL (1 = Scan 0 = Position) D0 = AZ (1 = Scan 0 = Position) If EL in position mode, maximum positive velocity is sent in the velocity field, if EL in scan mode, maximum or minimum elevation position is sent in the position field. If AZ in position mode, maximum positive velocity is sent in the velocity field, if AZ in scan mode, 0 is sent in the position field.
7	Control Word #2 D6 = Reset RCP02 on rising edge D5 = Noise Source On D4 = LSB of Pulse width D3 = Radiate On complemented D2 = Radiate On D1 = Servo Power On D0 = T/R Power On
8	Control Word #3 D6 = IRIS Mode 2 D5 = IRIS Mode 1 D4 = IRIS Mode 0 D3 = Radar Workstation A okay D2 = Radar Workstation B okay D1 = Data Processor A okay D0 = Data Processor B okay
9	Signal Generator level (0 ... 127 dB attenuation)
10	AZ Antenna Speed Low 7 bits
11	AZ Antenna Speed High 7 bits
12	EL Antenna Speed Low 7 bits
13	EL Antenna Speed High 7 bits
14	End Of Message (FF Hex)

Table 26 Status packet RCV03 format (RCP to host)

Char	Function
1	SYNC Byte (80 Hex)
2	Identification byte

Char	Function
3	Azimuth Low 7 bits (Earth relative)
4	Azimuth High 7 bits
5	Elevation Low 7 bits (Earth relative)
6	Elevation High 7 bits
7	Train Order Low 7 bits (azimuth of pedestal relative to the ship)
8	Train Order High 7 bits
9	Elevation Order Low 7 bits (elevation of pedestal relative to the ship)
10	Elevation Order High 7 bits
11	Pitch Low 7 bits
12	Pitch High 7 bits
13	Roll Low 7 bits
14	Roll High 7 bits
15	Heading Low 7 bits
16	Heading High 7 bits
17	Azimuth Rate Low 7 bits
18	Azimuth Rate High 7 bits
19	Elevation Rate Low 7 bits
20	Elevation Rate High 7 bits
21	Pitch Rate Low 7 bits (LSB = Zero)
22	Pitch Rate High 7 bits
23	Roll Rate Low 7 bits (LSB = Invalid Roll)
24	Roll Rate High 7 bits
25	Heading Rate Low 7 bits (LSB = Invalid Heading)
26	Heading Rate High 7 bits
27	Status #1 D6 = Low air flow D5 = Low Waveguide pressure D4 = Servo power D3 = Antenna Local mode D2 = Interlock open D1 = Standby D0 = Radiate ON

Char	Function
28	Status #2 D6 = RCP02 is shutdown D5 = LSB pulse width D4 = T/R Power on D3 = T/R Local mode D2 = Azimuth encoder calibrated D1 = MSB pulse width D0 = Magnetron current normal
29	Status #3 D6 = Reserved D5 = Reserved D4 = Reserved D3 = Elevation encoder calibrated D2 = Signal Generator fault D1 = Signal Generator On D0 = Signal Generator CW
30	Signal generator value (0=full signal)
31	Time Stamp Low 7 bits
32	Time Stamp High 7 bits
33	Latitude Low 7 bits
34	Latitude Middle 7 bits
35	Latitude High 7 bits
36	Longitude Low 7 bits
37	Longitude Middle 7 bits
38	Longitude High 7 bits
39	Altitude Low 7 bits
40	Altitude High 7 bits
41	Velocity East Low 7 bits (LSB = Invalid Lat/Lon)
42	Velocity East High 7 bits
43	Velocity North Low 7 bits (LSB = Zero)
44	Velocity North High 7 bits
45	Velocity Up Low 7 bits (LSB = Invalid Altitude)
46	Velocity Up High 7 bits
47	End Of Message (FF Hex)

Table 27 Status packet RCV05 format (RCP to host)

Char	Function
1–15	Bytes match the RCV02 / RCV04 format
16	Dual-System Status D6 = RCP02 is configured as a Dual-System D5 = Dual-System Mode MSB D4 = Dual-System Mode LSB D3 = This packet was sent from Unit "A"D2 = Information is known about the "Other" unit D1 = Unit "A" is the preferred system D0 = Unit "B" is disabled Note: The 2-bit Dual-System Mode codes are: • 00 : Unknown mode • 01 : System "A" • 10: System "B" • 11 : Auto Switch
17	Dual-System Status D6 = Unit "B" is okay D5 = Unit "B" Activity Code MSB D4 = Unit "B" Activity Code LSB D3 = Unit "A" is disabled D2 = Unit "A" is okay D1 = Unit "A" Activity Code MSB D0 = Unit "A" Activity Code LSB Note: The 2-bit Dual-System Activity codes are: • 00 : Inactive • 01 : Warmup • 10: Active Now • 11 : Reserved
18	Dual-System Status D6 = RCP02 is configured for voluntary flipping D5 = Unit "B" is offering to give up control D4 = Unit "A" is offering to give up control D3 = Unit "B" would be used if it were available D2 = Unit "A" would be used if it were available
19	D0:2 = Current Polarization XMT control • 0=Horizontal • 1=Vertical • 2=Alternating • 3=Simultaneous D3 = Polarization switch is OK to run
20	Spare
21	Spare
22	Spare

Char	Function
23	Spare
24	End Of Message (FF Hex)

Table 28 Control packet XMT05 format (host to RCP)

Char	Function
1–13	Bytes exactly match the XMT02 / XMT04 format
14	Control Word #4 D6 = Dual-System: Mode MSB D5 = Dual-System: Mode LSB D4 = Dual-System: Offer to relinquish control D3 = Dual-System: Unit would be used if available D2 = SpareD1 = Spare D0 = Spare Note: The 2-bit Dual-System Mode codes are: • 00 : No mode • 01 : System "A" • 10: System "B" • 11 : Auto Switch
15	D0: 2 = Requested Polarization XMT control 0=Horizontal 1=Vertical 2=Alternating 3=Simultaneous 7=Unchanged
16	Spare
17	Spare
18	End Of Message (FF Hex)

Table 29 Time packet (RCP to host)

Char	Function
1	SYNC Byte (B0 Hex)
2	Year Low 7 bits
3	Year High 7 bits
4	Month
5	Day
6	Hour

Char	Function
7	Minute
8	Second
9	1/100 of Second
10	Status
11	End Of Message (FF Hex)

Table 30 Generic BITE status packet (both ways)

Char	Function
1	SYNC Byte (C0 Hex)
2	BITE Unit ID byte (selectable in the range 00–7F Hex)
3	Status byte #1
4	Status byte #2
.	
.	
.	
N–1	Status byte #N–3
N	End Of Message (FF Hex)

The BITE status packet consists of 3 ...20 bytes.

The first 2 and the last bytes are used for identification purposes.

The middle bytes must have the MSB set to 0 and can contain arbitrary status in the lower 7 bits.

Typically this is used to report back individual bits containing the results of tests such as cabinet interlocks, airflow sensors, and power supply checks.

This report must be sent by the BITE every time the status changes. It must also send a report in response to the interrogate command. IRIS sends the interrogate command every 60 seconds.

Table 31 BITE command packet (both ways)

Char	Function
1	SYNC Byte (C0 Hex)
2	Command (0x4D = Interrogate, 0x44=Sample Data, 0x43=Reset)
3	End Of Message (FF Hex)

Auxiliary control BITE packets

This example BITE packet is supported by the RCP02 and RCP8 Radar Control Processors, and is an example of a mapping of arbitrary control and status bits into a BITE packet.

These RCPs contains 64 auxiliary status and control variables, labeled S[0:63] and C[0:63]. These bits may be sent to and from the host computer in the form of 13-byte BITE packets holding the full set of 64 bits. The format of these packets is the same in both directions, and the identification byte is selectable so that conflicts with other BITE packets can be avoided.

Table 32 Auxiliary control BITE packets (both ways)

Char	Function							
1	SYNC Byte (C0 Hex)							
2	BITE Unit ID byte (User Choice)							
3	Control/Status Bits	6	5	4	3	2	1	0
4	Control/Status Bits	13	12	11	10	9	8	7
5	Control/Status Bits	20	19	18	17	16	15	14
6	Control/Status Bits	27	26	25	24	23	22	21
7	Control/Status Bits	34	33	32	31	30	29	28
8	Control/Status Bits	41	40	39	38	37	36	35
9	Control/Status Bits	48	47	46	45	44	43	42
10	Control/Status Bits	55	54	53	52	51	50	49
11	Control/Status Bits	62	61	60	59	58	57	56
12	Control/Status Bit							63
13	End Of Message (FF Hex)							

Q-BITE Status

The Q-BITE (quantitative byte) status packets consists of 3 ... 128 bytes.

The first 2 and last bytes are for identification.

The middle bytes must have the MSB set to 0 and can contain an arbitrary value in the lower 7 bits. Typically this is used to report back voltage/power levels. Do not send this report by the BITE every time the status changes. Send this report in response to the Q-bite interrogate command. IRIS sends the interrogate command every 60 seconds.

The Q-BITE data stream consists of a series of integer values. Each value is packed into a series of 7-bit characters, using 1 ... 5 depending on the desired resolution. The low bits come first, and IRIS supports up to 32 bits per value. IRIS can be configured to display any such values with appropriate units and scaling.

Table 33 Q-BITE Status Packet (Both ways)

Char	Function
1	SYNC Byte (AF Hex)
2	BITE Unit ID byte (selectable in the range 00-7F Hex)
3	Status byte #1
4	Status byte #2
.	
.	
.	
N–1	Status byte #N–3
N	End Of Message (FF Hex)

Table 34 Simple Q-BITE example

Char	Function
1	SYNC Byte (AF Hex)
2	BITE Unit ID byte (Selectable in range 00–7F)
3	V1 Trigger frequency (Hz) (low bits)
4	V1 Trigger frequency (Hz) (high bits)
5	V2 Thyratron voltage (0.1 V) (low bits)
6	V2 Thyratron voltage (0.1 V) (high bits)
7	End Of Message (FF Hex)

Q-BITE interrogate packet

Table 35 Q-BITE interrogate packet (both ways)

Char	Function
1	SYNC Byte (90 Hex)
2	Command (0x01 or 0x4D=Interrogate, 0x44=Sample Data, 0x43=Reset)
3	End Of Message (FF Hex)

BITE individual command packet

The BITE individual command packet is used to request information about a single BITE unit, separate from all the others.

The RCP should respond to an **Interrogate** packet by sending the current version of the specified BITE status packet. The RCP should respond to a **Sample Data** packet by sending requests out the remote device to get information, then responding to the host computer with the new BITE status packet when the information arrives. The RCP should responds to the **Reset** packet by sending a reset command to the remote device.

Table 36 BITE individual command packet (host to RCP)

Char	Function
1	SYNC Byte (C1 Hex)
2	ID of the BITE unit for which the command will be applied
3	Command: 0x4D=Interrogate, 0x44=Sample Data, 0x43=Reset
4	End Of Message (FF Hex)

Chat mode

These packets are sent in both directions to convey serial TTY communication. Up to 6 7-bit characters can be sent in each packet with two characters of overhead for **SYNC** and **END**. This allows up to 75% of the available serial bandwidth to be used for chatting. If a **chat** mode packet contains fewer than 6 characters, then a NULL (0 byte) is inserted after the last packet.

Table 37 Chat mode packet (both ways)

Char	Function
1	SYNC Byte (F1 Hex)
2...7	7-Bit ASCII characters (possibly NULL terminated)
8	End Of Message (FF Hex)

Appendix C. Link transmission formats

C.1 AWS (Austrian Weather Service) format

The AWS (Austrian Weather Service) format uses an 8-bit data stream and can transmit an image of and rectangular product up to 256 pixels high with compression for horizontal runs of the same data.

For flexibility, IRIS supports a mapping table to convert the IRIS product color levels to the transmitted levels. Note the expanded color scales with more than 16 colors are not available for link outputs.



Here, hexadecimal notation is denoted with the **–h** suffix. For example, 12h means 12 base 16, or 18 base 10. Also for this discussion, all line and pixel numbers are origin 1. The upper left pixel is line 1, pixel 1.

The format of picture transmissions is as follows:

1. Station ID code
2. Time code
3. Data section
4. End-of-image code

The data section consists of a series of line number codes, followed by compressed data.

Data for a line ends when the next line number code is encountered. All unfilled pixels are assumed to be of the value 0. If a line is repeated in the data stream, any new data overwrites old data. If a line number code is never transmitted, the line is assumed to be filled with zeros.

The meaning of a byte is generally found by looking at the upper 4-bits. If those bits are a value between 0h and Eh, insert the data value in the low 4-bits 1 ... 15 times. If those bits are an **Fh**, it can be interpreted as one of the following special commands.

FFh — Line number follows in the next byte

The following byte contains one less than the line number for data that follows. For example, FFh 0Ch means line 13 follows. This command also implies that the previous line is zeroed from the current position to the end.

FEh — Extended data repeat command, information in next two bytes

This command is the same as the normal data repeat command, except that the repeat count is 12 bits. The high 8 bits of the repeat count are in the following byte, and the low 4 bits are in the high 4 bits of the third byte. The data value is in the low 4 bits of the third byte. For example, FEh 12h 34h means insert the data value 4 292 times.

FBh — Station identifier follows in next byte

The station identifier is configured from the **setup** utility. If multiple FBh commands are received for a given image, the last one takes effect. There must be at least one FBh command in each image. This command allows data from multiple sites to be merged on one transmission line. IRIS transmits only host site data.

FAh — Date and time follows in next 18 bytes

This command specifies the time of the data in the current image. If multiple FAh commands are received for a given image, the last one takes effect. There must be at least one FAh command in each image. The 18-byte string contains a text version of the time, such as "10-MAY-91 15:45:00." There is one space between the year and hours, and all numeric fields are two digits, with leading zeros, if required. The month contains the first three letters of the English month name.

F8h — End-of-Image

This indicates that an image has been completely sent. Any future transmissions relate to another image. After an F8h command and before an FFh command, only FBh, FAh commands are processed. All other commands are ignored.

C.2 Hong Kong Observatory format

The HKO (Hong Kong Observatory) format uses an 8-bit data stream and can transmit an image of and rectangular product up to 255 pixels high with compression for horizontal runs of the same data.

For flexibility, IRIS supports a mapping table to convert the IRIS product color levels to the transmitted levels. The expanded color scales with more than 16 colors are not available for link outputs.



Here, hexadecimal notation is denoted with the **–h** suffix. For example, 12h means 12 base 16, or 18 base 10. Also for this discussion, all line and pixel numbers are origin 1. The upper left pixel is line 1, pixel 1.

The format of picture transmissions is as follows:

1. Picture type ID code
2. Header code
3. Data section
4. End-of-image code

The data section consists of a series of line number codes, each followed by compressed data. Data for a line ends when the next line number code is encountered. All unfilled pixels are assumed to be of the value 1. If a line is repeated in the data stream, any new data overwrites old data. If a line number code is never transmitted, the line is assumed to be filled with ones.

The only use of a 0 byte is to immediately proceed on the following commands. This make it easy to sync with the data stream.

00h 01h — Picture type follows in the next byte

Table 38 HKO picture types

1	All PPIs			
2	CAPPI	>2.5 km height	<96 km range	data not velocity
3	CAPPI	all heights	<96 km range	data is velocity
4	CAPPI	<2.5 km height	<96 km range	data not velocity
5	CAPPI	>2.5 km height	96<range<192	data not velocity
6	CAPPI	all heights	96<range<192	data is velocity
7	CAPPI	<2.5 km height	96<range<192	data not velocity
8	CAPPI	all heights	>192 km range	data not velocity
9	CAPPI	all heights	>192 km range	data is velocity
10	All other products			

00h 02h — Header follows in the next 40 bytes

The format of the header is `HHPPPPPPPPPPPPhh:mm DD-MM-YYYY`. Where `HH` is the CAPPI height in km and `PPPPPPPPPPPP` is the product name.

All numeric fields are two digits, with leading zeros, if required.

00h 03h — Line number follows in the next byte

The following byte contains the origin 1 line number in the range 1–255. This is followed by a variable length string of bytes containing a data number followed by a repeat count. Data numbers are in the range 1 through 16.

00h 04h — End-of-Image

This indicates that an image has been completely sent. Any future transmissions relate to another image.

Appendix D. UF format

D.1 UF format overview

UF (Universal Format) is a radar data format originally proposed and documented in *Report on a Meeting to Establish a Common Doppler Radar Data Exchange Format*, page 1401 of the *November 1980 Bulletin of the American Meteorological Society*.

D.2 Single UF ray structure

The data is in a single file for a complete volume scan. The file includes a series of stand-alone rays (data acquired for a given pointing direction). Header information is duplicated for each ray.

Within a ray, data is organized as 16-bit words, byte swapped in the big endian convention. The exception is that each ray starts and ends with a 32-bit record size indicating the number of bytes in the ray.

ASCII text is usually left justified, space padded but some converter programs produce null terminated text.

The data format supports breaking large rays into several records. In this case, the multiple records within the ray have identical formats, with different field headers and data fields. The IRIS converter programs do not support this feature, and always place a ray in a single record. The optional header is placed in the ray for the first ray of a file.

The structure names usually end with a `2` to indicate that they are 2-byte aligned. By default, all IRIS structures are 4-byte aligned unless they end in a `2` or `8`.

ray size 4 Bytes
`<uf_mandatory_header2>` 45 Words
`<uf_optional_header>` 0 or 12 Words
`<uf_data_header2>` 3 + 2N Words
`<uf_field_header2>` #1 19 or 21 or 25 Words
data from field #1 M Words
`<uf_field_header2>` #2 19 or 21 or 25 Words
data from field #2 M Words
...(repeats for each data type)...
ray size 4 Bytes

For more information, see the *`uf.h`* header file.

D.3 uf_data_header2 structure

Source: `uf.h`

Byte	Size	Contents
0	int16_t	Number of fields in this ray
2	int16_t	Number of records in this ray
4	int16_t	Number of fields in this record
6	int16_t	Data type of field #1 (standard): `VR:velocity SW:spectrum width DR:ZDRCZ:Corrected dBZ DZ:Total dBZ RH:RhoHVPH:PhiDP KD:KDP LH:LdrHLV:LdrV`
8	int16_t	Field #1 field header position
10	int16_t	Data type of field #2
12	int16_t	Field #2 field header position
...		

D.4 uf_field_header2 structure

Source: `uf.h`

Byte	Size	Contents
0	int16_t	Data offset from start of record, origin 1
2	int16_t	Scale factor, met units = file value/scale
4	int16_t	Start range km
6	int16_t	Start range meters
8	int16_t	Bin spacing in meters
10	int16_t	Bin count
12	int16_t	Pulse width in meters
14	int16_t	Horizontal beam width in degrees*64
16	int16_t	Vertical beam width in degrees*64
18	int16_t	Receiver bandwidth in Mhz*64 ?
20	int16_t	Polarization: 1:horz 2:vert 3:circular 4:elliptical
22	int16_t	Wave length in cm*64

Byte	Size	Contents
24	int16_t	Sample size
26	char[2]	Type of data used to threshold
28	int16_t	Threshold value
30	int16_t	Scale
32	char[2]	EditCode
34	int16_t	PRT in microseconds
36	int16_t	Bits per bin, must be 16
38	12	`<uf_fsi2>`

D.5 uf_fsi2 structure

Source: **`uf.h`**

Byte	Size	Contents
If velocity data:		
0	int16_t	Nyquist velocity
2	int16_t	`<spare>`
If DM data:		
0	int16_t	Radar Constant
2	int16_t	Noise Power
4	int16_t	Receiver Gain
6	int16_t	Peak Power
8	int16_t	Antenna Gain
10	int16_t	Pulse Duration (microseconds*64)
If other data:		
nothing		

D.6 uf_mandatory_header2 structure

Source: **`uf.h`**

Byte	Size	Contents
0	`char[2]`	Text **UF**
2	`int16_t`	Record Size in 16-bit words

Byte	Size	Contents
4	`int16_t`	Offset to start of optional header, origin 1
6	`int16_t`	Local-Use Header Position (origin 1)
8	`int16_t`	Data Header Position (origin 1)
10	`int16_t`	Record Number (origin 1)
12	`int16_t`	Volume number on tape, n/a for disk
14	`int16_t`	Ray number within the volume scan
16	`int16_t`	Record number within ray (origin 1)
18	`int16_t`	Sweep number within the volume scan
20	`char[8]`	Radar name
28	`char[8]`	Site name
36	`int16_t`	Latitude degrees (North positive, South negative)
38	`int16_t`	Latitude minutes
40	`int16_t`	Latitude seconds*64
42	`int16_t`	Longitude degrees (East positive, West negative)
44	`int16_t`	Longitude Minutes
46	`int16_t`	Longitude Seconds
48	`int16_t`	Height of antenna above sea level in meters
50	`int16_t`	Year (time of data acquisition)
52	`int16_t`	Month
54	`int16_t`	Day
56	`int16_t`	Hour
58	`int16_t`	Minute
60	`int16_t`	Second
62	`char[2]`	Time zone, "UT" for universal
64	`int16_t`	Azimuth (degrees*64) of midpoint of sample
66	`int16_t`	Elevation (degrees*64)
68	`int16_t`	Sweep mode:
		0:Cal 1:PPI 2:Coplane 3:RHI

Byte	Size	Contents
		4:Vertical 5:Target 6:Manual 7:Idle
70	`int16_t`	Fixed angle (degrees*64)
72	`int16_t`	Sweep rate ((degrees/second)*64)
74	`int16_t`	Year (generation data of UF format)
76	`int16_t`	Month
78	`int16_t`	Day
80	`char[8]`	Name of UF generator program
88	`int16_t`	Value stored for deleted or missing data (0x8000)

D.7 uf_optional_header structure

Source: `uf.h`

Byte	Size	Contents
0	char[8]	sProjectName[8]
8	int16_t	iBaselineAzimuth
10	int16_t	iBaselineelevation
12	int16_t	iVolumeScanHour /* Time of start of current volume scan */
14	int16_t	iVolumeScanMinute
18	char[8]	sFieldTapeName[8]
24	int16_t	iFlag

Appendix E. RTD format

E.1 Real Time Display overview

IRIS can output a real-time display (RTD) data stream using UDP data packets. You can use one of those provided or use your own.

The source code for the transmitter programs is in *base/rtq_lib*. The formats are documented in the *sig_rtdisp.h* file.

Any computer running IRIS on that network can receive and display the real time display data.

Because real time display is done in broadcast mode, it is scalable. If the IRIS Radar server sends real time display data, the load induced on the network, and the servers CPU is constant regardless of how many machines on the network receive and process the real time display data.

UDP data is not retransmitted upon packet loss. At the application level in IRIS, no attempt is made for detected packet losses nor are there any requests for retransmissions. If such a loss occurs at the transmitter, on the network, or at one or more receiving hosts, then a gap occurs in the picture drawn on the destination real time displays. This differs from IRIS product transmission which are done point-to-point via TCP and are reliable due to error checking and retransmissions.

The network load from a real time display server broadcasting is a function of the following:

Table 39 Network load from RTD dependencies

Dependency	Description
Number of moments being broadcast	Up to 3. Including **Z**, **V**, and **W**.
Number of bins per radial	The total number of bins, combined for all moments, is approximately 1400 per radial. There is an overhead of about 100 bytes per ray.
Scan rate of the antenna	Each bin requires one byte of data. A scan rate of 5 RPM (30 radials per second), requires about 1500 * 30 = 45000 bytes per second. Real time display bandwidth must be added to any other IRIS and other bandwidth being occupied on your network.

The real time display data stream can be configured to transmit using up to 6 different formats, IP addresses, and ports. See the **Setup** utility documentation in *IRIS and RDA Utilities Guide*.

Normally, gateways such as multi-homed hosts, bridges and routers do NOT pass IRIS real time display data. This is because such devices are built to filter broadcast data. This keeps the real time display data on a single network. However, it is usually possible to configure such devices to pass broadcast data that occurs on a specific port (like the real time display port).

To receive real time display data, the receive process needs to issue a socket and a bind system call. Following this, the receiver can read messages from the socket as they are transmitted.

E.2 Rtd_nids3_xmt

This format is designed to support a legacy data format used by Radtec.

All packets are the same format, with minimal header information.

Data can be either 8-bit or 4-bit format.

The config file *`rtd_nids3_xmt.conf`* controls some details.

E.3 Rtd_v1_xmt

Three messages are transmitted as part of the real time display data stream. The messages are distinguished by the first 2 bytes (`SHORT`) in the message summarized in the list table (see `sig_rtdisp.h`):

- `RTRAY_TYPE_HEADER`
- `RTRAY_TYPE_RAY`
- `RTRAY_TYPE_RAY`

The message `RTRAY_TYPE_HEADER` consists of the 2-byte ID followed by a `struct rtd_v1_vol_header`. This message is sent either at the beginning of a new elevation sweep, or periodically if a new sweep has not started recently.

The message `RTRAY_TYPE_RAY` begins with the 2-byte ID. What follows depends on the type of data being transmitted by the real time display sender. This is chosen in the sender's **setup** utility and can consist of 1, 2, or 3 data types being Z, V, or W. The volume header indicates which data types are currently transmitted and in which order the data types are presented.

For example, if only Z and V are transmitted, the `RTRAY_TYPE_RAY` message consists of its 2-byte message ID, followed by a struct `rtd_v1_ray_header`, followed by the data elements for Z, followed by the V elements. The Z and V elements are presented as one byte per range bin.

For example, if the volume header indicates that 200 bins of each type are transmitted, then the Z elements (and the V elements) are each 200 bytes long, with each byte representing 1 range bin. The spacing between the range bins is also defined by the volume header.

For information on element scaling, see Data types (page 71).

E.4 Rtd_v2_xmt

This format is designed to support the full number of range bins, almost the full number of data types, and both 8-bit and 16-bit data formats. As a result each radial must be split into potentially many individual 1500 byte UDP packets.

The volume header `rtd_v2_vol_header` is sent periodically. Every 25 rays, or when there is a change. Otherwise the `rtd_v2_ray_header` is sent followed by data.

Appendix F. Supported IRIS input pipes

IRIS uses a general input pipe mechanism that allows users to import data from other meteorological systems for combination with radar data.



Source code is provided for most input pipes to allow users to modify them for format changes or to create pipes for new input data.
For information on each pipe, see the corresponding configuration file.

The **Syntax** column in the tables below indicates which command-line format a pipe expects. For more information, see Input pipes (page 17).

Latest input pipes

The table below lists the new `ngpipes`-generation converters. The naming follows the new `…2iris` naming style.

Table 40 Latest input pipes

Name	Supported product types	Supported data types	Syntax	Purpose
`cfrnc2iris`	`RAW` (polar; PPI and RHI scans)	T, Tv, Z, Zv, Zc, V, W, SQI, ZDR, PhiDP, KDP, RhoHV, LDRh, LDRv, SNR, CSR	Pathnames	Converts CfRadial 2.0 polar data (PPI/RHI scans; other scan types rejected) to an **IRIS RAW** product. *Note: frozen and largely superseded by the `cfrnc2pb` protobuf path.*
`grib2iris`	`PPI, MAX, VIL, RAIN1`	Z, V, W, VIL, Rain Rate	Pathnames	Converts WMO GRIB (edition 1/2) fields to an IRIS product.
`hdf52iris`	`RAW, BASE, PPI, CAPPI, TOPS, HMAX, MAX, RAIN1, RAINN, RHI, SRI, VVP, VIL, XSECT`	All polar moments (RAW); per-product (Cartesian)	Pathnames	Converts an ODIM HDF5 file to an IRIS product. Supports latest (2.4) ODIM version.
`metar2iris`	`SWS` (surface weather station)	Surface observations (temperature, wind, and so on)	Pathnames	Converts METAR/SYNOP surface-station reports to an **IRIS SWS** product.

Legacy input pipes

The table below lists the original SIGMET-style converters. The naming follows the legacy `…ToIris` / `…To…` naming style.

Table 41 Legacy input pipes

Name	Supported product types	Supported data types	Syntax	Purpose
`AsciiToGage`	`GAGE`	Rain rate (rain gauge)	Pipe	Converts ASCII rain gauge/ disdrometer data to an **IRIS GAGE** product.
`AsciiToPlane`	`WARN`	None (track)	Pipe	Converts ASCII aircraft-position data to an **IRIS WARN** overlay.
`BMPSatToIris`	`USER`	None (image RGB/ greyscale)	Pathnames	Converts a BMP satellite/ background image to an **IRIS USER** image product.
`BufrToIris`	`PPI, CAPPI, TOPS, MAX, RAIN1, RAINN, SRI, VIL, VVP`	Z, V, W, ZDR, LDRh, VIL, Rain Rate, RAIN, H	Pathnames	Converts WMO BUFR Cartesian products (OPERA guidelines) to IRIS. Raw data and Lambert azimuthal equal area projection are not supported.
`Dwr2ToIris`	`RAW`	T, Z, V, W, ZDR, RhoHV, PhiDP	Pathnames	Converts ISRO DWR-2 (DWR MK-II) polar data to an **IRIS RAW** product.
`IMDSatToIris`	`SATELLITE`	None (satellite)	Pathnames	Converts India Meteorological Department (IMD) satellite HDF5 data to an IRIS satellite product.
`InmToIris`	`RAW`	Z	Pipe	Converts INM polar-volume data to an **IRIS RAW** product.
`KmaRadToIris`	`CAPPI`	Rain rate	Pipe	Converts a KMA gridded radar array to an **IRIS CAPPI** product.
`KmaSatToIris`	`USER`	None (image)	Pipe	Converts a KMA satellite image to an **IRIS USER** image product.
`KnmiHDF5ToIris`	`SATELLITE`	None (satellite/ image)	Pathnames	Converts KNMI HDF5 data to an `IRIS satellite` product.
`McIdasSatToIris`	`USER`	None (image)	Pipe	Converts McIDAS area-file imagery to an **IRIS USER** product (SeaLaunch).
`UfToIris`	`RAW`	T, Z, Zc, V, W, SQI, ZDR, PhiDP, KDP, RhoHV, LDRh, LDRv	Pathnames	Converts Universal Format (UF) polar data to an **IRIS RAW** product.

Appendix G. Supported IRIS output pipes

The following tables show the output pipes supplied with IRIS.

Using standard Linux commands to convert to PDF is also supported.

The **Syntax** column in the tables below indicates which command-line format a pipe expects. For more information, see Input pipes (page 17).

Latest output pipes

The newer `ngpipes`-generation converters (`iris2…` naming).

Table 42 Latest output pipes

Name	Supported product types	Supported data types	Syntax	Purpose
`iris2ascii`	`RAW`, all Cartesian products (`CAPPI, PPI, MAX, RAIN1, RAINN, SRI, TOPS, BASE, HMAX, LAYER, RHI, XSECT, USER, USERV, VIL`) and non-Cartesian products (`VVP, VAD, GAGE, TRACK, CATCH, FCAST, NDOP, SLINE, STATUS, WIND`)	All moments (RAW); per-product otherwise	Pathnames	Converts an IRIS product to Octave-readable ASCII.
`iris2csv`	`RAW, CAPPI, PPI, MAX, RAIN1, RAINN, SRI, TOPS, BASE, HMAX, LAYER, RHI, XSECT, USER, USERV, VIL`	All moments (RAW); per-product (Cartesian)	Pathnames	Converts an IRIS product to CSV.
`iris2grib`	`PPI, MAX, VIL, RAIN1`	Z, V, W, VIL, Rain Rate	Pathnames	Converts an IRIS product to WMO GRIB edition 1 or 2 (eccodes).
`iris2kml`	`CAPPI`	None (georeferences a rendered image)	Pathnames	Converts an IRIS product to a georeferenced KML/KMZ overlay (image via `--im`).
`iris2mfarome`	`PPI`	Z, V	Pathnames	Converts IRIS PPI products to Météo-France AROME BUFR.

Name	Supported product types	Supported data types	Syntax	Purpose
`iris2mfpolar`	`RAW` (polar)	Z, ZDR, PhiDP, RhoHV	Pathnames	Converts IRIS polar sweeps to Météo-France polarimetric BUFR.
`iris2mfzraster`	`PPI, CAPPI`	Z	Pathnames	Converts IRIS reflectivity rasters to Météo-France raster BUFR.
`iris2odimhdf5`	`RAW, BASE, PPI, CAPPI, TOPS, HMAX, MAX, RAIN1, RAINN, RHI, SRI, VVP, VIL, XSECT`	T, Z, V, W, SQI, ZDR, ZDRc, PhiDP, KDP, RhoHV, LDRh, LDRv, HCLAS, VIL, Rain Rate, H, SNR	Pathnames	Converts IRIS products to ODIM HDF5 (OPERA guidelines). Support latest (2.4) version.
`iris2xml`	`RAW, BASE, PPI, CAPPI, TOPS, HMAX, MAX, RAIN1, RAINN, RHI, SRI, VVP, VIL, XSECT`	T, Z, V, W, SQI, ZDR, ZDRc, PhiDP, KDP, RhoHV, LDRh, LDRv, HCLAS, VIL, Rain Rate, H, SNR	Pathnames	Converts IRIS products to ODIM-style XML.

Legacy output pipes

The original SIGMET-style converters (older **`IrisTo…`** naming).

Table 43 Legacy output pipes

Name	Supported product types	Supported data types	Syntax	Purpose
`IrisToAscii`	`RAIN1, RAINN, CAPPI, PPI, TOPS, BASE, MAX, VIL, SRI, THICK, LAYER, HMAX`	Any (no data-type filter; Rain Rate/RAIN read as 16-bit, others as 8-bit)	Pipe	Converts a Cartesian product to ASCII (lat/lon/value) text. Older pipe; ships in the **`miscio`** package together with the **`sig_ftp`**/**`sig_lftp`**/**`sig_sftp`** transfer scripts used in IRIS output-device configuration.
`IrisToAsterix`	`CAPPI, PPI, SRI`	T, Z, Zc	Pipe	Converts an IRIS product to Eurocontrol ASTERIX format.
`IrisToBufr`	`PPI, CAPPI, RAIN1, RAINN, VIL, SRI, TOPS, BASE, HMAX, RHI, VVP, MAX, RAW`	Z, Zc, V, W, ZDR, LDRh, LDRv, Rain Rate, RAIN, VIL, H, HCLAS	Pipe	Converts IRIS products to WMO BUFR (OPERA guidelines), with optional GTS headers. RAW products are encoded as separate reflectivity/velocity/width messages.

Name	Supported product types	Supported data types	Syntax	Purpose
`IrisToEwis`	`RAW, CAPPI, MAX, HMAX, TOPS, RAIN1, RAINN`	T, Z, Zc, V, Vc, W, ZDR, H	Pipe	Converts IRIS products to Ericsson EWIS format.
`IrisToMcidas`	`BASE, CAPPI, HMAX, MAX, PPI, RAIN1, RAINN, SHEAR, SRI, TOPS, VIL`	Z, Zc, V, Vc, W, ZDR, LDRh, LDRv, VIL, Rain Rate, RAIN, H, SHEAR	Pathnames	Converts IRIS products to McIDAS area files.
`IrisToNetCDF`	`RAW`	All RAW moments (T, Z, Zc, V, Vc, W, SQI, ZDR, ZDRc, KDP, PhiDP, RhoHV, LDRh, LDRv, SNR, HCLAS, PMI, LOG, CSR, ...)	Pathnames	Converts an IRIS RAW product to NetCDF.
`IrisToUf`	`RAW`	T, Z, Zc, V, Vc, W, SQI, ZDR, ZDRc, PhiDP, KDP, RhoHV, LDRh, LDRv	Pathnames	Converts an IRIS RAW product to Universal Format (UF).

Appendix H. Common-DTO protobuf pipes

These pipes convert to and from the IRIS Focus / common DTO **protobuf (PB)** format.

…2pb converters produce protobuf, and the **pb2…** converters consume it. These all are actively developed.

The **Syntax** column in the tables below indicates which command-line format a pipe expects. For more information, see Input pipes (page 17).

Table 44 Common DTO protobuf pipes

Name	Supported product types	Supported data types	Syntax	Purpose
`cfrnc2pb`	**RAW** (polar / Level 2; PPI, RHI and DBS sweeps)	T, Tv, Z, Zv, Zc, V, W, SQI, ZDR, PhiDP, KDP, RhoHV, LDRh, LDRv, SNR, CSR	Pathnames	Converts CfRadial 2.0 polar data to IRIS Focus protobuf (also handles lidar moments).
`iris2pb`	**RAW, PPI, CAPPI, MAX, RAIN1, RAINN, SRI, TOPS, BASE, HMAX, LAYER, VIL, RHI, XSECT, SHEAR, USER, USERV, SATELLITE**	All RAW moments present	Pathnames	Converts an IRIS RAW/ Cartesian product to IRIS Focus protobuf.
`odimhdf52pb`	**RAW** (polar)	All polar moments present	Pathnames	Converts ODIM HDF5 polar data to IRIS Focus protobuf.
`pb2cfrnc`	**RAW** (polar)	All RAW polar moments	Pathnames	Converts IRIS Focus protobuf to CfRadial 2.0 NetCDF.
`pb2geotiff`	Raster (Cartesian) products only	Single selectable moment (for example, Z)	Pathnames	Converts IRIS Focus protobuf to a GeoTIFF image (single raster moment).
`pb2iris`	**RAW** (volumes), Cartesian raster products, **GAGE**	All RAW moments present (volumes); raster data (products); rain accumulation (**GAGE**)	Pathnames	Converts IRIS Focus protobuf back to an IRIS RAW, Cartesian or GAGE product. Vector/event payloads are not supported.
`pb2odimhdf5`	**RAW** (polar)	All polar moments present	Pathnames	Converts IRIS Focus protobuf to ODIM HDF5.

Appendix I. Shell-script wrappers for conversion pipes

In addition to the pipe binaries, the following shell-script wrappers are available in the same directory as the pipes:

- **`iris2cfrnc.sh`** chains **`iris2pb`** → **`pb2cfrnc`** to convert IRIS data to CfRadial 2.0 NetCDF format.
- **`iris2geotiff.sh`** chains **`iris2pb`** → **`pb2geotiff`** to convert IRIS data to GeoTIFF format.

Warranty

For warranty terms and conditions and the warranty claim process, see vaisala.com/warranty.

Note that warranty coverage may not apply if the product is damaged due to normal wear and tear, exceptional operating conditions, negligent handling or installation, or unauthorized modifications. See the applicable supply contract or Conditions of Sale for details of the warranty for each product.

Technical support



You can contact Vaisala technical support at vaisala.my.site.com/myvaisala or helpdesk@vaisala.com. You can find all our support channels at vaisala.com/support.

When contacting technical support, provide the following as applicable:

- Product name, model, and serial number
- Software/Firmware version
- Name and location of the installation site
- Name and contact information of a technical person who can provide further information
- Diagnostics package for the application

If applicable, also provide the following details:

- What failed (what worked / did not work)?
- Where did it fail (location and environment)?
- When did it fail (date, immediately / after a while / periodically / randomly)?
- How many failed (only one defect / other same or similar defects / several failures in one unit)?
- What was done when the failure was noticed?
- What was connected to the product and to which connectors?
- What is the input power source type, voltage, and what else (such as lighting, heaters, and motors) was connected to the same power output?
- Are all parts connected and grounded properly? Take a photo to help the troubleshooting.

Recycling



Recycle all applicable material according to local regulations.

VAISALA