User Manual

```
 0:  00 00 00 60 00 00 00 00  00 00 00 00 00 00 00 00   ...`............
10:  3D C3 2C A0 00 00 00 00  00 00 00 00 C0 27 54 05   =Ã,.........À'T.
20:  BD C8 B4 39 3C D8 C5 18  3B F8 2F 92 44 B5 C0 00   ½È´9<ØÅ.;ø/´DµÀ.
30:  44 B3 7E CD 00 00 00 00  BD 9E 70 8A BE 14 CB C4   D³~Í....½▌p▌¾.ËÄ
40:  3F DD 40 88 00 00 00 00  BD 9E 70 8A BE 14 CB C4   ?Ý@▌....½▌p▌¾.ËÄ
50:  3F DD 40 88 00 00 24 00  41 7D 2C 2C 3D EF BD B1   ?Ý@▌..$.A},,=ï½±
60:  C2 26 E5 D9 00 00 24 00  41 7E 16 8C 3D F0 A0 EB   Â&åÙ..$.A~.▌=ð ë
70:  C2 23 C9 3C 00 00 24 00  41 80 11 8C 3D F2 9D 77   Â#É<..$.A▌.▌=ò▌w
80:  C2 21 7F FE 00 00 24 00  41 80 8D 31 3D F3 8D 35   Â!▌þ..$.A▌▌1=ó▌5
90:  C2 1E 99 49 00 00 24 00  41 80 AF A9 3D F3 D0 0B   Â.▌I..$.A▌¯©=óÐ.
A0:  C2 1B 5C 61 00 00 24 00  41 81 29 C1 3D F4 BC C8   Â.\a..$.A▌)Á=ô¼È
B0:  C2 18 A3 63 00 00 24 00  41 82 0F FC 3D F6 7B 33   Â.£c..$.A▌.ü=ö{3
C0:  C2 16 7F 88 00 00 24 00  41 82 F0 D5 3D F8 2F 2C   Â.~▌..$.A▌ðÕ=ø/,
```

```
SURF Nr Of Events              : 0
SURF Nr Of Polygonelements     : 0


Sounding Nr 0   Profiletime: '13:33:40.265'
  Heading: 210.20 deg Heave: -0.10 m Roll:  1.52 deg Pitch:  0.43 deg
  Fullfan
  CenterPos X:      491444 m  Y:    5878435 m
    Beam 000  Depth:     15.8 PosAhead:      0.1 PosAstar:   -41.7  X:     491480 m  Y:    5878414 m
    Beam 001  Depth:     15.9 PosAhead:      0.1 PosAstar:   -40.9  X:     491479 m  Y:    5878414 m
    Beam 002  Depth:     16.0 PosAhead:      0.1 PosAstar:   -40.4  X:     491479 m  Y:    5878414 m
    Beam 003  Depth:     16.1 PosAhead:      0.1 PosAstar:   -39.6  X:     491478 m  Y:    5878415 m
    Beam 004  Depth:     16.1 PosAhead:      0.1 PosAstar:   -38.8  X:     491478 m  Y:    5878415 m
    Beam 005  Depth:     16.1 PosAhead:      0.1 PosAstar:   -38.2  X:     491477 m  Y:    5878415 m
    Beam 006  Depth:     16.3 PosAhead:      0.1 PosAstar:   -37.6  X:     491477 m  Y:    5878416 m
    Beam 007  Depth:     16.4 PosAhead:      0.1 PosAstar:   -37.1  X:     491476 m  Y:    5878416 m
    Beam 008  Depth:     16.5 PosAhead:      0.1 PosAstar:   -36.5  X:     491476 m  Y:    5878416 m
```

ATLAS SURF API-3.1

The Software Interface for ATLAS Echo-Sounders

Table of Content

1 Change Log

Version	Changes	Author(s)	Date
0.9 english	First Draft	S. Könnecke	June, 28th 2003
0.91 english	Corrections and Complements	J. Brinkmann	July, 07th 2003
1.0 english	First Final	S. Könnecke	July, 14th 2003

2 Introduction

SURF ("Sensor-Unabhaengiges Rohdaten-Format") is a widely used ATLAS format for exchange of hydro acoustic data. The SURF format contains data of one specific hydro-acoustic sensor accompanied by assigned data of all other sensors that are relevant for hydrographic data processing. SURF is used as a transfer format from the ATLAS HYDROMAP ONLINE to hydrographic data processing systems like the ATLAS HYDROMAP OFFLINE.

SURF data are generated by the ATLAS HYDROMAP ONLINE or by the SURF API using the system independent LAN transfer format XDR. The SURF format supports parallel data access (for fast memory access) and serial data transfer.

SURF data are transferred via LAN or data exchange media to data processing tools or user-dependent application programs for interpretation and transformation into other data output formats.

The SURF format is the same for all sensors of the same sounder type (multi-beam echo-sounder or single-beam echo-sounder). SURF includes sensor data information of relevant sensors like position sensors, gyros and motion sensors. The sensor data information is assigned to the hydro acoustic data of the sounding sensor. The sensor data information is interpolated for each sounding according to time (for transmit time and receive time of the acoustic pulse). The SURF format contains other data necessary for data processing as well (e.g. water sound velocity profiles, tide measurements). Data which are valid for a number of soundings are stored only once.

SURF data are accessible from external side via the SURF API leading to the following customer benefit: Changes in the SURF format do not require changes in customer application software; they only require the inclusion of an updated version of the SURF API. The SURF API is a function library written in ANSI C. The provided SURF API source code is compatible with Unix and Windows platforms and was tested on HP-UX, Sun-OS, Dec-Unix, Sgi-Irix, Linux, Windows 9x, Windows NT and Windows 2000.

Over the years, the SURF Format has been extended twice:

- The original SURF format was SURF 1.
- The extension to the SURF 2 format includes backscatter information.
- The extension to the SURF 3 format includes TPE information. (TPE = Total Propagated Error)

The current SURF API is able to read all types of SURF files (SURF 1, SURF 2 and SURF 3) and writes SURF 3 files.

3 SURF Data Format

In the ATLAS SURF data format the data is organized in profiles and soundings. Each SURF file set consists of a .SDA and a .SIX :file with the same root name. Each SURF file set represents exactly one survey profile (e.g. a planned survey line, a covered area).

SURF data is recorded by ATLAS vertical or multi-beam echo-sounders or by ATLAS multi-channel vertical echo-sounders. Every time a ping is transmitted into the water a sounding record is created. That means that the echoes of one ping of one vertical sounder is exactly one sounding and so are the echoes of a multi-beam system. For a vertical echo-sounder this would be one single or for a dual-frequency sounder two single depths per sounding. For a multi-beam echo-sounder the whole swath of echoes received by the transducer is stored into one sounding.

Each sounding has its own characteristics like for instance position, time and sound velocities.

3.1 Data Extraction Process

The SURF data format is intended for usage in combination with the SURF API. However, it is possible to extract some more detailed information by accessing raw data records through low level SURF API functions (discussed in chapter 0).

3.2 Sounding Depend Data Sets

Sounding accompanying data sets are stored in different tables and referenced by time or index:

- Angle data sets (SurfDataInfo, SurfMultiBeamAngleTable)
- Transducer parameter (SurfDataInfo, SurfTransducerParameterTable)
- Sound velocity profiles (SurfDataInfo, SurfCProfileTable)
- Manual and cyclic events (SurfDataInfo, SurfEvents)
- The indices for the transducer, sound velocity and angle tables are located in the area "SurfDataInfo, SdaInfo, SurfSoundingData". There are additional elements for each sounding:

- the motion values and heading at transmission time,
- the mean sound velocity and the velocity at the transducer face,
- the tide value and
- the time and way relative to the profile start.

3.3 Depth and Offset Calculation

Through SURF-API traveltimes of each beam of each swath are accessible. Depth values of each sounding can be calculated based on these values.

ATLAS multi-beam echo-sounders apply active roll stabilisation to the data. The ATLAS HYDROSWEEP DS/DS2 does also apply active pitch stabilisation.

Each reception beam angle at the transducer face is roll corrected online during acquisition based on real-time data of the connected motion sensor. The echo-sounder internal angle table is constant and therefore kept independently from vessel roll motion. The beam angle is defined as the angle from a apparent vertical perpendicular to sea-surface (see Fig. 1) crossing the beam at the transducers centre point. The traveltimes are stored with respect to that angle of the (constant) angle table that equals the roll corrected beam angle at transducer face.

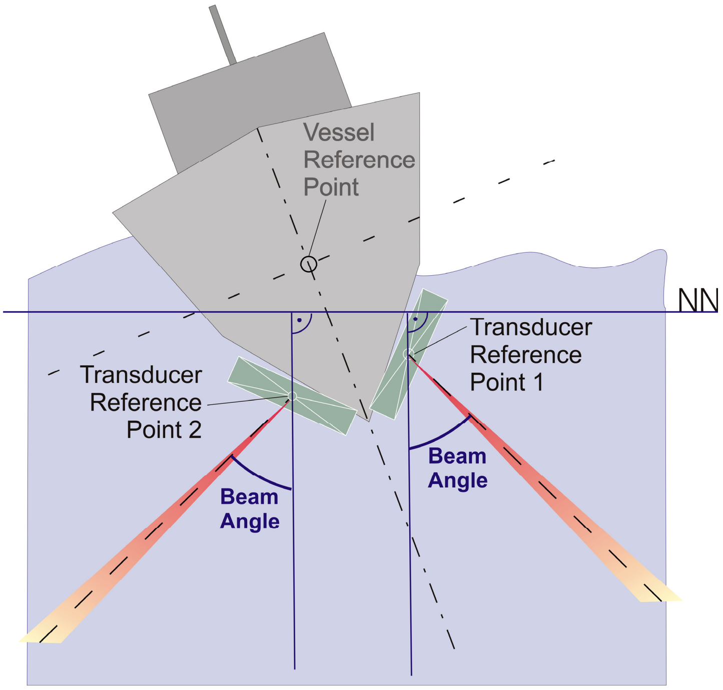

Fig. 1 Beam Angle Reference

The function **`depthFromTT`**, stored in library/pb_math.c, will give an example how to calculate the depth from the traveltime. The depth is already heave and draught corrected.

`depthFromTT` uses the following algorithm:

- known:
  - TT: travel time,
  - α: beam angle from angle table,
  - cmean: mean sound velocity through the water column,
  - cmean: sound velocity at the transducer face,
  - πTX: pitch during transmission,
  - ηTX, ηRX: heave during transmission (TX) and reception (RX)
  - δ: transducer draught
- $s = TT \cdot c_{mean} - \frac{(\eta_{TX} - \eta_{RX})}{2} \cos \overline{\alpha}$; slant range,
- $\overline{\alpha} = \arcsin\left(\frac{c_{mean}}{c_{keel}} \sin(\alpha) \right)$; sv corrected angle,
- $\Delta h = \frac{s}{\sqrt{\tan^2(\overline{\alpha}) + \tan^2(\pi) + 1}}$; pitch corrected depth, without draught and heave,
- $\begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} = \begin{pmatrix} \Delta h \cdot \tan(\pi) + x_{offset} \\ \Delta h \cdot \tan(\overline{\alpha}) + y_{offset} \end{pmatrix}$; position ahead (Δx) and astar (Δy)
- $h = \Delta h - \eta + \delta$; heave and draught corrected depth.

4 The Surf API

The SURF API library provides a set of functions, which allow access to SURF data in a convenient way. This version 3.1.5 is able to read all SURF data up to version SURF 3 (actual). If new SURF format versions are introduced, the SURF API will be updated. For version control the function **`void SAPI_printAPIandSURFversion(void)`** will give an output of the current library version to **`stderr`**.

4.1 Structure of the Surf API

The SURF API is a library enabling easy access to SURF data.

One of the source code files is the header file sapi.h describing the contents of the different SURF modules and declaring the SURF API function prototypes. The directory 'libsrc' includes the source code of the SURF API library.

In the directory 'examples', you find two example source files (sapitest1.c, sapitest2.c) showing how to read data and using most of the SURF API functions. There are three additional examples which show how to build your own SURF data files (sapitest3.c, sapitest4.c, sapitest5.c).

In the directory 'data', you find two different SURF files for testing the examples or testing your own application.

In the directory 'doc', you find a description of the API functions.

For compilation of the SURF API, an ANSI C compiler is required.

Two additional system libraries must be linked to your application: - the math-lib 'libm.a' - the library with the xdr-routines

In order to build the library and the example programs on a specific Unix system, please adapt the Makefiles in the directory 'libsrc' and in the directory 'examples' first: uncomment the compile and link options for the used Unix system and use make or gmake (on DEC-Unix) from the main directory. For Windows systems, you can find the project definition files for Visual Studio 6.0 in the following directories: example/win_sapitest1, example/win_sapitest2, example/win_sapitest4 and libsrc/libsapi

4.2 File Handling and Management Functions

The surf API can only handle one SURF data set once in a while.

long SAPI_open(char* surfDir, char* surfFile, long errorprint)

...opens the management of a SURF file and gives access to the first sounding.

Parameters in : surfDir is the path to SURF data

surfFile is the name of the SURF file

(without '.xxx' extension)

errorprint = 0 suppresses error prints

errorprint <> 0 gives error prints on stderr

out: returnvalue = 0 means 'open successful'

returnvalue<> 0 look for error print

long SAPI_nextSounding(long errorprint)

...gives access to the next sounding. This function moves the SURF API file reader one sounding forward. So following operation will affect the next sounding.

Parameters in: errorprint = 0 suppresses error prints

errorprint <> 0 gives error prints on stderr

out: returnvalue = 0 means 'move successful'

returnvalue<> 0 error or EOF

long SAPI_rewind(long errorprint)

...this function rewinds the whole profile. After rewinding the SURF API file reader will point to the first sounding.

Parameters in : errorprint = 0 suppresses error prints

errorprint <> 0 gives error prints on stderr

out: returnvalue = 0 means 'move successful'

returnvalue<> 0 error

void SAPI_close(void)

...closes the SURF data files of the actual loaded SURF file set and cleans memory allocations.

4.3 Data from SIX Index File

The '.six' file contains global data and reference tables. It is used to organise the comprehensive data stored in the .SDA file.

4.3.1 Meta Data

char* SAPI_getNameOfShip(void)

...returns the ship's name

Parameters out: returnvalue = "?" means 'data not available'

char* SAPI_getNameOfSounder(void)

...returns the sounder name

Parameters out: returnvalue = "?" means 'data not available'

char* SAPI_getTypeOfSounder(void)

...returns the sounder type

Parameters out: returnvalue = "?" means 'data not available'

"M" means MANUAL_DATA

"D" means DIGITIZED_DATA

"V" means VERTICAL_SOUNDER (e.g. ATLAS DESO 25)

"B" means BOMA_TYPE_SOUNDER (multi channel echo-sounder)

"F" means FAN_TYPE_SOUNDER (multi-beam echo-sounder),

e.g. ATLAS FANSWEEP 20/15)

SurfGlobalData* SAPI_getGlobalData(void)

...returns a pointer to the SurfGlobalData object . This objects holds global information for the echo-sounder such as lever arms and projection. ATLAS Software (e.g. ATLAS Hydromap Online) allows the input of such information before survey. Most of this information are extracted out of the configuration files (*.ini)

Parameters out: returnvalue = NULL means 'data not available'

SurfGlobalData Structure

Item	Type	Comment
label	char [LABEL_SIZE]	
shipsName	char [STRING_SIZE]	Name of the ship on which this data set is recorded
startTimeOfProfile	char [TIME_SIZE]	Time/Date of the first swath
regionOfProfile	char [STRING_SIZE]	'N','E','S','W'
numberOfProfile	char [STRING_SIZE]	The raw data profile name
chartZero	Float	Customers chart vertical reference
tideZero	Float	Tide vertical reference
numberOfMeasuredSoundings	u_long	Msd. Soundings in this profile
actualNumberOfSoundingSets	u_long	Stored soundings in this profile
timeDateOfTideModification	char [TIME_SIZE]	Time/Date of modifications, made with the ATLAS HYDROMAP OFFLINE Editor
timeDateOfDepthModification	char [TIME_SIZE]	...
timeDateOfPosiModification	char [TIME_SIZE]	...
timeDateOfParaModification	char [TIME_SIZE]	...
correctedParameterFlags	u_long	Flags set by SURF conversion or Editor if raw values have been modified: 1 = tide corrected 2 = draught corrected 4 = course manipulated 8 = heave manipulated 16 = roll manipulated 32 = pitch manipulated 64 = ckeel manipulated 128 = cmean manipulated 256 = slope corrected 512 = reduced raw data used 1024 = squat corrected 2048 = heave compensated
offsetHeave	Float	The motion sensors heave offset
offsetRollPort	Float	Not used
offsetRollStar	Float	Not used
offsetPitchFore	float	Not used
offsetPitchAft	float	Not used
nameOfSounder	char [STRING_SIZE]	echo-sounder name
typeOfSounder	char	"M" = MANUAL_DATA "D" = DIGITIZED_DATA "V" = VERTICAL_SOUNDER "B" = BOMA_TYPE_SOUNDER "F" = FAN_TYPE_SOUNDER
highFrequency	float	> 70 kHz
mediumFrequency	float	> 15kHz...70kHz
lowFrequency	float	<= 15kHz
nameOfEllipsoid	char [STRING_SIZE]	projection ellipsoid name
semiMajorAxis	double	projection ellipsoid parameter
flattening	double	projection ellipsoid parameter
projection	char [STRING_SIZE]	map projections type used for positioning

Item	Type	Comment
presentationOfPosition	char	‘E’ = lat[°]/long[°]/height[m] or ‘X’ = X[m]/Y[m]/height[m]
referenceMeridian	double	for (Transverse) Mercator projections (e.g. GK, UTM)
falseEasting	double	Projections false easting
falseNorthing	double	Projections false northing
referenceOfPositionX	double	Projection origin
referenceOfPositionY	double	Projection origin
presentationOfRelWay	char	‘l’
planedTrackStartX	float	Planed track information
planedTrackStartY	float	...
planedTrackStopX	float	...
planedTrackStopY	float	...
originalTrackStartX	float	Surveyed track information
originalTrackStartY	float	...
originalTrackStopX	float	...
originalTrackStopY	float	...
originalStartStopDistance	float	The Distance between track start and stop point [m]
originalStartStopTime	double	The time, which has past surveying from start to stop
timeDateOfTrackModification	char [TIME_SIZE]	
modifiedTrackStartX	float	Information of the modified track
modifiedTrackStartY	float	...
modifiedTrackStopX	float	...
modifiedTrackStopY	float	...
modifiedStartStopDistance	float	...

SurfEventValues SAPI_getEvent(long nrEvent)*

...returns a pointer to the SurfEventValues objects

Parameters in : nrEvent = the number of Event

0..(SAPI_getNrEvents()-1)

Parameters out: returnvalue = NULL means 'data not available'

SurfEventValue Structure

Item	Type	Comment
positionX	double	Position of Event
positionY	double	Position of Event
relTime	float	Time of Event
text	char [EVENT_SIZE]	Event string (e.g. "Saint John Harbour")

SurfPolygons SAPI_getPolygons(void)*

...returns a pointer to the SurfPolygons object

Parameters out: returnvalue = NULL means 'data not available'

SurfPolygon Structure

Item	Type	Comment
label	char [LABEL_SIZE]	‘POLYGON’
values	SurfPolygonValues [1]	ptr to a dynamically allocated point array

SurfPolygonValues Structure

Item	Type	Comment
polygonX	double	Polygon point X co-ordinate
polygonY	double	Polygon point Y co-ordinate

4.3.2 Data Organisation

long SAPI_getNrSoundings(void)

...returns the number of stored soundings of this SURF file set (i.e. this profile).

long SAPI_getNrBeams(void)

...returns the number of beams per sounding. The number of beams my differ from sounding to sounding. `SAPI_getNrBeams()` delivers the maximal number of beams per sounding of the whole profile.

long SAPI_getNrSoundvelocityProfiles(void)

...tells how many sound velocity profiles have been stored within this profile

long SAPI_getNrEvents(void)

...tells how many events have been stored within this profile

long SAPI_getNrPolygonElements(void)

...tells how many polygon points are describing the area covered by this profile. Polygons are not stored by ATLAS sensors. Polygons can be defined by the user and stored into self made SURF data sets e.g. for planning purposes.

long SAPI_dataHaveHighFrequencyLayer(void)

...returns information, whether the file contains data of the high frequency layer (> 70kHz)

Parameters out: returnvalue

= 1 means HF data are available

= 0 means HF data are not available

long SAPI_dataHaveMediumFrequencyLayer(void)

...returns information, whether the file contains data of the medium frequency layer (> 15kHz...70kHz)

Parameters out: returnvalue

= 1 means MF data are available

= 0 means MF data are not available

long SAPI_dataHaveLowFrequencyLayer(void)

...returns information, whether the file contains data of the medium frequency layer (<= 15kHz)

Parameters out: returnvalue

= 1 means LF data are available

= 0 means LF data are not available

4.3.3 External Reference - Time, Position, Sound Velocity

To be able to combine echo-sounder data with external sensors such as GPS receivers, a accurate time reference is essential.

double SAPI_getAbsoluteStartTimeOfProfile(void)

...converts absolute times in SURF TIME_SIZE presentation into a double format which allows mathematics on times (adding. relative times in SURF data etc.). The resolution of this double is higher than 1 second. Typecasting this double to 'time_t' will give a standard C time presentation (loosing the fractions of a second).

The positioning information of the position sensors is transformed in the defined map projection. The position of the first sounding is stored in absolute ellipsoid co-ordinates either in [rad] or in [meter]. The positions of all following soundings are stored in [meter] relative to the position of the first sounding.

long SAPI_posPresentationIsRad(void)

...returns the scaling of the centerPosition.

Parameters out: returnvalue

= 1 means scaling is [rad]

= 0 means scaling is [m]

long SAPI_getNrPositionsensors(void)

...tells how many position sensors have been stored. The first sensor is that one, which has been selected as the system sensor during data acquisition.

SurfPositionAnySensor* SAPI_getPositionSensor(long nrSensor)

...returns a pointer to the SurfPositionAnySensor objects of sensor number nrSensor. If the label of positioning sensor is 'POLARFIX', the returned SurfPositionAnySensor should be casted to an SurfPositionPolarFix object so that the specific Polarfix items can be accessed.

Parameters in: nrSensor = the number of the position sensor

0..(SAPI_getNrPositionsensors()-1)

Parameters out: returnvalue = NULL means 'data not available'

SurfPositionAnySensor Structure

Item	Type	Comment
label	char [LABEL_SIZE]	- 'UNKNOWN','INTEGRATED NAV','SYLEDIS','MNS2000','GPS' or 'EPIRB' - for the 'POLARFIX' sensor a special object structure (SurfPositionPolarfix) is available
positionSensorName	char [STRING_SIZE]	Name of the positioning sensor
none1	Float	not used
none2	Float	not used
none3	Float	not used
none4	Float	not used
none5	Float	not used
none6	Float	not used
none7	Float	not used
none8	Float	not used
time9	char [TIME_SIZE]	time stamp of positioning
none10	Float	not used
none11	Float	not used
none12	Float	not used
none13	Float	not used
none14	Float	not used
none15	Float	not used
none16	Float	not used
none17	Float	not used
sensorAntennaPositionAhead	Float	lever arm of positioning sensor
sensorAntennaPositionStar	Float	lever arm of positioning sensor
sensorAntennaPositionHeight	Float	lever arm of positioning sensor

SurfPositionPolarfix Structure

Item	Type	Comment
label	char [LABEL_SIZE]	- should be 'POLARFIX' - otherwise the structure SurfPositionAnySensor should be used
positionSensorName	char [STRING_SIZE]	Name of the positioning sensor
polarfixLocationX	float	
polarfixLocationY	float	
polarfixLocationZ	float	
polarfixReferenceX	float	
polarfixReferenceY	float	
polarfixReferenceZ	float	
polarfixReferenceDistance	float	
polarfixReferenceAngle	float	
timeOfLastPolarfixEdit	char [TIME_SIZE]	time stamp of positioning
polarfixEditLocationX	float	
polarfixEditLocationY	float	
polarfixEditLocationZ	float	
polarfixEditReferenceX	float	
polarfixEditReferenceY	float	
polarfixEditReferenceZ	float	
polarfixEditReferenceDistance	float	
polarfixEditReferenceAngle	float	
polarfixAntennaPositionAhead	float	

Item	Type	Comment
polarfixAntennaPositionStar	float	
polarfixAntennaPositionHeight	float	

SurfStatistics* SAPI_getStatistics(void)

...returns a pointer to the SurfStatistics object . The SurfStatistics object holds minimum and maximum of all external sensors for the whole profile (i.e. for the whole SURF data set). For instance it can be used to put this profile in a global geo-reference context.

Parameters out: returnvalue = NULL means 'data not available'

SurfStatistics Structure

Item	Type	Comment
label	char [LABEL_SIZE]	
minNorthing	Double	Minimum and maximum of co-ordinates define a bounding rectangle of all data within this survey profile (SURF data set).
maxNorthing	Double	
minEasting	Double	
maxEasting	Double	
minSpeed	Float	Minimum and maximum speed during this profile
maxSpeed	Float	
minRoll	Float	The limits of the motion sensor data give a good estimate of the sea state and i.e. the reliability and accurcy of measured depths.
maxRoll	Float	
minPitch	Float	
maxPitch	Float	
minHeave	Float	
maxHeave	Float	
minBeamPositionStar	float	The beam position limits describe the covered area and is used to for the automatic scaling of the editor.
maxBeamPositionStar	Float	
minBeamPositionAhead	Float	
maxBeamPositionAhead	Float	
minDepth	float	minimum depth in this profile
maxDepth	float	maximum depth in this profile

4.4 Data From SDA Mass Data File

The '.SDA' file contains sounding dependent mass data in a strict sequential manner. Opening a SURF file set offers access to the first element of that sequence. The user can access the following records by calling `SAPI_nextSounding()`.

4.4.1 Sounding Data

SurfSoundingData* SAPI_getSoundingData(void)

...returns a pointer to the actual SurfSoundingData object

Parameters out: returnvalue = NULL means 'data not available'

SurfSoundingData Structure

Item	Type	Comment
soundingFlag	u_short	The sounding/swath description: 1 = sounding is deleted 2 = course manipulated 4 = heave manipulated 8 = roll manipulated 16 = pitch manipulated 32 = ckeel manipulated 64 = cmean manipulated 128 = slope corrected 256 = split fan (not set: full fan) if 256 is set: 512 = starboard fan (not set: port fan) 1024 = ahead fan (ds2 calibration, not set: normal fan) 2048 = all beams are deleted
indexToAngle	u_short	Index of the related angle table within the angle table array
indexToTransducer	u_short	Index of the related transducer description within the transducer array
indexToCProfile	u_short	Index of the related sound velocity profile within the sound velocity profile array
relTime	float	time stamp of this sounding relative to SAPI_getGlobalData->originalStartStopTime().
relWay	float	travelled way from start of profile on
tide	float	tide value of this sounding
headingWhileTransmitting	float	
heaveWhileTransmitting	float	
rollWhileTransmitting	float	
pitchWhileTransmitting	float	
cKeel	float	sound velocity measured at transducer face at the time of transmission (if keel-svp available).
cMean	float	mean sound velocity in the water column. this is generated from the related sound velocity profile (Cprofile).
dynChartZero	float	chart zero for this sounding

4.4.2 Sounding Data References

SurfTransducerParameterTable* SAPI_getActualTransducerTable(void)

...returns a pointer to the in the actual sounding indexed SurfTransducerParameterTable defining e.g. the position of the transducer in the ship's reference co-ordinate system .

Parameters out: returnvalue = NULL means 'data not available'

SurfTransducerParameterTable Structure

Item	Type	Comment
label	char [LABEL_SIZE]	'TRANSDUCERTABLE'
transducerDepth	float	The transducers Z – value (ref.: Water Level)
transducerPositionAhead	float	Ahead difference to reference point
transducerPositionStar	float	Starboard difference to reference point
transducerTwoThetaHFreq	float	
transducerTwoThetaMFreq	float	
transducerTwoThetaLFreq	float	

SurfMultiBeamAngleTable* SAPI_getActualAngleTable(void)

...returns a pointer to the in the actual sounding indexed SurfMultiBeamAngleTable

Parameters out: returnvalue = NULL means 'data not available'

SurfMultiBeamAngleTable Structure

Item	Type	Comment
label	char [LABEL_SIZE]	
actualNumberOfBeams	u_short	Number of beams in this sounding
beamAngle	float [1]	a dynamically allocated array of beam angles (actualNumberOfBeams long).

SurfCProfileTable* SAPI_getActualCProfileTable(void)

...returns a pointer to the in the actual sounding indexed SurfCProfileTable

Parameters out: returnvalue = NULL means 'data not available'

SurfCProfileTable Structure

Item	Type	Comment
label	char [LABEL_SIZE]	
relTime	float	relative time of sound velocity profile
numberOfActualValues	u_short	number of sv values
values	CprofileValues [1]	dyn. allocated array of sv values

CprofileValues Structure

Item	Type	Comment

Item	Type	Comment
depth	Float	depth of logged velocity
cValue	Float	logged (measured) sound velocity

SurfCenterPosition* SAPI_getCenterPosition(long nrPositionSensor)

...returns a pointer to the SurfCenterPosition object of the actual swath.

Parameters in : nrSensor = the number of PositionSensor

0..(SAPI_getNrPositionsensors()-1)

Parameters out: returnvalue = NULL means 'data not available'

SurfCenterPosition Structure

Item	Type	Comment
positionFlag	u_short	1 = position deleted
centerPositionX	Float	Relative position X
centerPositionY	float	Relative position Y
speed	float	Ship's speed

4.4.3 Bathymetric Data

SurfSingleBeamDepth* SAPI_getSingleBeamDepth(void)

...returns a pointer to the actual SurfSingleBeamDepth object, if the sounding represents a vertical echo-sounder measurement. Otherwise NULL will be returned.

Parameters out: returnvalue = NULL means 'data not available'

SurfSingleBeamDepth Structure

Item	Type	Comment
depthFlag	u_short	1 = deleted 2 = object 4 = fraction line 8 = manual data input 16 = tide corrected 32 = tide is manipulated 64 = position is manipulated 128 = high frequency depth is manipulated 256 = medium frequency depth is manipulated 512 = low frequency depth is manipulated 1024 = draught corrected 2048 = depth is suppressed
travelTimeOfRay	float	Not used
depthHFreq	float	Depth of the high frequency transducer
depthMFreq	float	Depth of the medium frequency transducer
depthLFreq	float	Depth of the low frequency transducer

SurfMultiBeamDepth* SAPI_getMultiBeamDepth(long beam)

...returns a pointer to the actual SurfMultiBeamDepth objects of the selected beam, if the sounding represents a multi-beam echo-sounder measurement. Otherwise NULL will be returned.

The retrieved depths are already corrected for heave, roll, pitch, yaw and sound velocity cmean.

Parameters in: beam = the number of selected beam

0..(SAPI_getNrBeams()-1)

Parameters out: returnvalue = NULL means 'data not available'

SurfMultiBeamDepth Structure

Item	*Type*	*Comment*
depthFlag	u_short	1 = deleted 2 = object 4 = fraction line 8 = manual data input 16 = tide corrected 32 = tide is manipulated 64 = position is manipulated 128 = depth is manipulated 1024 = draught corrected 2048 = depth is suppressed 4096 = the fan is reduced 8192 = On looking for the transducer increase the current index by 1.
depth	float	The calculated depth
beamPositionAhead	float	Position of the beam ahead (relative to the center position
beamPositionStar	float	Position of the beam starboard (relative to the center position)

SurfMultiBeamTT SAPI_getMultiBeamTraveltime(long beam)*

...returns a pointer to the actual SurfMultiBeamTT objects of the selected beam if the sounding represents a vertical echo-sounder measurement. Otherwise NULL will be returned. These times represent one way travel time. The selected beam is one out of the (user-) pre-defined beam angle table. It is a reception beam corrected for roll and sound velocity at transducer face (see 3.3 for more details on this).

Parameters in : beam = the number of selected beam

0..(SAPI_getNrBeams()-1)

Parameters out: returnvalue = NULL means 'data not available'

SurfMultiBeamTT Structure

Item	*Type*	*Comment*
travelTimeOfRay	float	in seconds. This is the one way travel time from transducer face to bottom.

SurfMultiBeamReceive SAPI_getMultiBeamReceiveParams(long beam)*

...returns a pointer to the actual SurfMultiBeamReceive objects of the selected beam

Parameters in: beam = the number of selected beam

0..(SAPI_getNrBeams()-1)

Parameters out: returnvalue = NULL means 'data not available'

SurfMultiBeamReceive Structure

Item	Type	Comment
headingWhileReceiving	float	Heading on the beams receiving time
heaveWhileReceiving	float	Heave on the beams receiving time

4.4.4 Sidescan & Backscatter Related Data

SurfAmplitudes* SAPI_getMultibeamBeamAmplitudes(long beam)

...returns a pointer to the actual SurfAmplitudes objects of the selected beam

Parameters in: beam = the number of selected beam

0..(SAPI_getNrBeams()-1)

Parameters out: returnvalue = NULL means 'data not available'

SurfAmplitudes Structure

Item	Type	Comment
beamAmplitude	u_short	Amplitude of the beam

SurfExtendedAmplitudes* SAPI_getMultibeamExtendedBeamAmplitudes(long beam)

...returns a pointer to the actual SurfExtendedAmplitudes objects of the selected beam

Parameters in: beam = the number of selected beam

0..(SAPI_getNrBeams()-1)

Parameters out: returnvalue = NULL means 'data not available'

SurfExtendedAmplitudes Structure

Item	Type	Comment
mtau	float	Time interval for beam amplitude access
nis	u_short	Noise isotropic value
beamAmplitude	u_short	Average amplitude of the beam

SurfSignalParameter* SAPI_getMultibeamSignalParameters(void)

...returns a pointer to the actual SurfSignalParameter object (TVG data,etc. : see sapi.h)

Parameters out: returnvalue = NULL means 'data not available'

SurfSignalParameter Structure (new since version 2.2)

Item	Type	Comment
bscatClass	u_short	Id of the classification from the backscatter editor
nrActualGainSets	u_short	Number of stored gain sets
rxGup	Float	Transformation value pressure to volt of receive transducer
rxGain	Float	Actual gain value on receive
ar	Float	Frequency dependent attenuation
rxSets	TvgRxSets []	array of TVG curve steps

TvgRxSets Structure

Item	Type	Comment
time	Float	scale: sec
gain	float	scale: dB

SurfTxParameter SAPI_getMultibeamTransmitterParameters(void)*

...returns a pointer to the actual SurfTxParameter object (TX power,etc.)

Parameters out: returnvalue = NULL means 'data not available'

SurfTxParameter Structure

Item	Type	Comment
txSets[1]	TxSets []	Specification of the different beam pattern

txSets Structure

Item	Type	Comment
txBeamIndex	u_long	Code of external beamshapetable
txLevel	float	Transmission level, scale : dB rel 1 uPa
txBeamAngle	float	Transmission angle, scale : rad
pulseLength	float	Transmission pulse length, scale : sec

SurfSidescanData SAPI_getSidescanData(void)*

...returns a pointer to the actual SurfSidescanData object

Parameters out: returnvalue = NULL means 'data not available'

SurfSidescanData Structure

Item	Type	Comment
sidescanFlag	u_long	Not used
actualNrOfSsDataPort	u_short	Number of sidescan data on port side
actualNrOfSsDataStb	u_short	Number of sidescan data on starboard side
minSsTimePort	float	Minimum time on port
minSsTimeStb	float	Minimum time on Starboard
maxSsTimePort	float	Maximum time on port
maxSsTimeStb	float	Maximum time on starboard
ssData	u_char [1]	dyn allocated array of sidescan data values (amplitudes); first Port then Stb

4.5 Straightforward Functions to Access Basic Sounding Data

These functions calculate XYZ co-ordinates for depth values of the actual sounding based on simple processing algorithms. These functions maybe slower than plain code based on plain SURF Data Records.

The resulting XYZ co-ordinates are draft, heave pitch, roll and yaw corrected. There is also a correction for sound velocity at the transducer interface applied. This does not include any ray tracing through the water column. Of course, all correction depend on the availability of the corresponding external sensor data (i.e. motion sensor, SVP).

Horizontal reference for X and Y is the vessels reference point. The vertical reference is defined as NN. Therefore heave and transducer draught are already taken care of. It is also possible to access chartZero-referenced depth values by setting function input parameter `depthOverChartZero` to be <> 0.

4.5.1 Multi-beam Data Access

long SAPI_getXYZfromMultibeamSounding(long nrBeam, long depthOverChartZero, double* north, double* east, double* depth)

...returns depth and absolute position of the selected beam in the actual sounding

Parameters in: nrBeam = the number of selected beam

0..(SAPI_getNrBeams()-1)

depthOverChartZero

= 0 means depth over normal zero

= 1 means depth over during data acquisition defined chart zero

north,east,depth = adresses for the result

Parameters out: returnvalue

= 0 means 'data valid"

<> 0 means 'data not available' or 'data deleted or suppressed"

4.5.2 Single-Beam Data Access

ATLAS vertical echo-sounders have one or more frequency channels. All frequency channels are stored within one profile (i.e. within one SURF data set). Different frequency layers can be discriminated by a 3 different frequency flags: LF, MF and HF.

LF < 15kHz < MF < 70 kHz < HF

long SAPI_getXYZfromSinglebeamSoundingHF(long depthOverChartZero, double* north, double* east, double* depth)

...returns depth and absolute position of the the actual sounding (HF layer)

Parameters in: depthOverChartZero

= 0 means depth over normal zero

= 1 means depth over during data acquisition defined chart zero

north,east,depth = adresses for the result

Parameters out: returnvalue

= 0 means 'data valid'

<> 0 means 'data not available' or 'data deleted or suppressed'

long SAPI_getXYZfromSinglebeamSoundingMF(long depthOverChartZero, double* north, double* east, double* depth);

...returns depth and absolute position of the the actual sounding (MF layer)

Parameters in: depthOverChartZero

= 0 means depth over normal zero

= 1 means depth over during dataaquisition defined chart zero

north,east,depth = adresses for the result

Parameters out: returnvalue

= 0 means 'data valid'

<> 0 means 'data not available' or 'data deleted or suppressed'

long SAPI_getXYZfromSinglebeamSoundingLF(long depthOverChartZero, double* north, double* east, double* depth)

...returns depth and absol. position of the the actual sounding (LF layer)

Parameters in: depthOverChartZero

= 0 means depth over normal zero

= 1 means depth over during dataaquisition defined chart zero

north,east,depth = adresses for the result

Parameters out: returnvalue

= 0 means 'data valid'

<> 0 means 'data not available' or 'data deleteed or suppressed'

4.6 Functions for Writing Your Own SURF Data

For instance for software simulation or survey planning purposes you can create your own SURF files. The following chapter gives a short introduction in corresponding SURF API functions.

long SAPI_openIntoMemory(char* surfDir,char* surfFile,long errorprint)

...creates an empty SURF file set and initialises some basic data structures to handle the data in RAM. The function gives access to the first sounding. The whole data set is presented in memory, you may modify it and write it back to files. This function call may access a high amount of memory. SAPI_nextSounding and SAPI_rewind are working as well on this kind of opening SURF data.

Parameters in: surfDir is the path to SURF data

surfFile is the name of the SURF file (without '.xxx' extension)

errorprint = 0 suppresses error prints

errorprint <> 0 gives error prints on stderr

out: returnvalue

= 0 means 'opened successful'

returnvalue<> 0 look for error print

long SAPI_createSurfBody(long nrSoundings, long nrBeams, long maxNrSidescanSamplesPerSounding, long errorprint)

...creates an empty SURF body for a couple of simple configurations which you may fill with your own data sets. The empty data structures are presented in memory, you may fill them and write them back to files. This function call may access a high amount of memory. SAPI_nextSounding and SAPI_rewind are working as well on this kind of opening SURF data.

Parameters in: nrSoundings tells how many soundings have to be stored nrBeams tells on how many beams each sounding has been received

=0 means that the data come from a vertical sounder.

maxNrSidescanSamplesPerSounding tells how many sidescan

samples have to be stored atworst case per sounding

= 0 means, there are no sidescan data at all .

errorprint

= 0 suppresses error prints

<> 0 gives error prints on stderr

out: returnvalue

= 0 means 'created successful'

<> 0 look for error print

long SAPI_writeBackFromMemory(char surfDir,char* surfFile,long errorprint)*

...writes back SURF data from memory to files that have been opened with SAPI_openIntoMemory or that have been build by SAPI_createSurfBody.

Parameters in: surfDir is the path to SURF data

surfFile is the name of the SURF file (without '.xxx' extension)

surfDir and surfFile maybe different then during opening.

errorprint

= 0 suppresses error prints

<> 0 gives error prints on stderr

out: returnvalue

= 0 means 'written successful'

<> 0 look for error print

This collection of functions might be extended in later versions of the SURF API.

5 Copyright and Licensing

This SURF API does not come with any warranties, nor is it guaranteed to work on your computer or to do anything useful. The user assumes full responsibility for the use of this library.

6 Index