SURF API Version 3.1.4

The SURF API library provides a set of functions, which allow access to SURF data in a convenient way. This version 3.1.4 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.

File Handling and Management Functions

```
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
   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);

...moves back 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 files and cleans memory allocations



Data from SIX Index File
------------------------

 The '.six' file contains global data and reference tables

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
                                  "B" means BOMA_TYPE_SOUNDER
                                  "F" means FAN_TYPE_SOUNDER


long SAPI_getNrSoundings(void);

...returns the number of stored soundings


long SAPI_getNrBeams(void);

...returns the number of beams per 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 aquisition


long SAPI_getNrSoundvelocityProfiles(void);

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


long SAPI_getNrEvents(void);

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


long SAPI_getNrPolygonElements(void);

...tells how many polygon points are describing
   the area covered by this profile



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.)
   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).



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



SurfGlobalData* SAPI_getGlobalData(void);

...returns a pointer to the SurfGlobalData object (see sapi.h)
   Parameters out:  returnvalue = NULL means 'data not available'



SurfStatistics* SAPI_getStatistics(void);

...returns a pointer to the SurfStatistics object (see sapi.h)
   Parameters out:  returnvalue = NULL means 'data not available'



SurfPositionAnySensor* SAPI_getPositionSensor(long nrSensor);

...returns a pointer to the SurfPositionAnySensor objects (see sapi.h)
   Parameters in :  nrSensor    = the number of the position sensor
                                  0..(SAPI_getNrPositionsensors()-1)
   Parameters out:  returnvalue = NULL means 'data not available'



SurfEventValues* SAPI_getEvent(long nrEvent);

...returns a pointer to the SurfEventValues objects (see sapi.h)
   Parameters in :  nrEvent     = the number of Event
                                  0..(SAPI_getNrEvents()-1)
```

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



SurfPolygons* SAPI_getPolygons(void);

...returns a pointer to the SurfPolygons object (see sapi.h)
   Parameters out:  returnvalue = NULL means 'data not available'



Data From SDA Mass Data File
----------------------------

 The '.sda' file contains sounding dependent mass data


SurfSoundingData* SAPI_getSoundingData(void);

...returns a pointer to the actual SurfSoundingData object (see sapi.h)
   Parameters out:  returnvalue = NULL means 'data not available'



SurfTransducerParameterTable* SAPI_getActualTransducerTable(void);

...returns a pointer to the in the actual sounding indexed
               SurfTransducerParameterTable
   Parameters out:  returnvalue = NULL means 'data not available'


SurfMultiBeamAngleTable* SAPI_getActualAngleTable(void);

...returns a pointer to the in the actual sounding indexed
               SurfMultiBeamAngleTable
   Parameters out:  returnvalue = NULL means 'data not available'


SurfCProfileTable* SAPI_getActualCProfileTable(void);

...returns a pointer to the in the actual sounding indexed
               SurfCProfileTable
   Parameters out:  returnvalue = NULL means 'data not available'



SurfCenterPosition* SAPI_getCenterPosition(long nrPositionSensor);
```

```
...returns a pointer to the actual SurfCenterPosition objects (see sapi.h)
   Parameters in :  nrSensor    = the number of PositionSensor
                                  0..(SAPI_getNrPositionsensors()-1)
   Parameters out:  returnvalue = NULL means 'data not available'



SurfSingleBeamDepth* SAPI_getSingleBeamDepth(void);

...returns a pointer to the actual SurfSingleBeamDepth object (see sapi.h)
   Parameters out:  returnvalue = NULL means 'data not available'



SurfMultiBeamDepth* SAPI_getMultiBeamDepth(long beam);

...returns a pointer to the actual SurfMultiBeamDepth objects of the
   selected beam (see sapi.h)
   Parameters in :  beam        = the number of selected beam
                                  0..(SAPI_getNrBeams()-1)
   Parameters out:  returnvalue = NULL means 'data not available'



SurfMultiBeamTT* SAPI_getMultiBeamTraveltime(long beam);

...returns a pointer to the actual SurfMultiBeamTT objects of the
   selected beam (see sapi.h)
   Parameters in :  beam        = the number of selected beam
                                  0..(SAPI_getNrBeams()-1)
   Parameters out:  returnvalue = NULL means 'data not available'



SurfMultiBeamReceive* SAPI_getMultiBeamReceiveParams(long beam);

...returns a pointer to the actual SurfMultiBeamReceive objects of the
   selected beam (see sapi.h)
   Parameters in :  beam        = the number of selected beam
                                  0..(SAPI_getNrBeams()-1)
   Parameters out:  returnvalue = NULL means 'data not available'



Sidescan & Backscatter Related Data from SDA Mass Data File
-----------------------------------------------------------
```

```
SurfAmplitudes* SAPI_getMultibeamBeamAmplitudes(long beam);

...returns a pointer to the actual SurfAmplitudes objects of the
   selected beam (see sapi.h)
   Parameters in :  beam        = the number of selected beam
                                  0..(SAPI_getNrBeams()-1)
   Parameters out:  returnvalue = NULL means 'data not available'



SurfExtendedAmplitudes* SAPI_getMultibeamExtendedBeamAmplitudes(long beam);

...returns a pointer to the actual SurfExtendedAmplitudes objects of the
   selected beam (see sapi.h)
   Parameters in :  beam        = the number of selected beam
                                  0..(SAPI_getNrBeams()-1)
   Parameters out:  returnvalue = NULL means 'data not available'



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'



SurfTxParameter* SAPI_getMultibeamTransmitterParameters(void);

...returns a pointer to the actual SurfTxParameter object
   (TX power,etc. : see sapi.h)
   Parameters out:  returnvalue = NULL means 'data not available'



SurfSidescanData* SAPI_getSidescanData(void);

...returns a pointer to the actual SurfSidescanData object (see sapi.h)
   Parameters out:  returnvalue = NULL means 'data not available'



Simple Functions to Access Basic Sounding Data
----------------------------------------------
```

```
   simple routines, but maybe slower than plain code on pointered objects

  /* depthOverChartZero =  0 --> depth over normal 0
     depthOverChartZero =  1 --> depth over in SURF defined Chartzero */

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
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'


  ---different Frequencylayers may be stored in one profile of vertical
sounders---
                  --- 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
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_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'



Functions for Writing Your Own SURF Data
----------------------------------------

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

...opens the management of a SURF file and 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 datastructures 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 (pings into the
                                  water) have to be stored .
                    nrBeams       tells on how many beams each sounding
                                  has been received . (nrBeams=0 means
                                  that the data come from a vertical
                                  sounder).
                    maxNrSidescanSamplesPerSounding  tells how many
                                  sidescan samples have to be stored at
                                  worst case per sounding
                                  (0 means, there are no sidescan data
                                   at all) .
                    errorprint  = 0 suppresses error prints
                    errorprint <> 0 gives error prints on stderr
              out:  returnvalue = 0 means 'created successful'
                    returnvalue<> 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
                    errorprint <> 0 gives error prints on stderr
              out:  returnvalue = 0 means 'written successful'
                    returnvalue<> 0 look for error print



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