RFC: Reference Manual Entries for VDS functions

Neil Fortner, Quincey Koziol, Elena Pourmal

This document proposes *HDF5 Reference Manual* (RM) entries for the creation functions that support the Virtual Dataset (VDS) feature in HDF5.

Table of Contents

1 Introduction

As we develop the VDS feature, we will be adding new public functions to the library. This document summarizes RM entries for the new functions. Each section of the document corresponds to a specific VDS function.

We used the same style for API names, parameters, and return values as for the existing functions with similar functionality such as `H5Pset_external`, `H5Pset_external_count`, and `H5Iget_name`.

The document will evolve during the lifespan of the project. It will be updated each time the team starts working on new VDS functionality. The document will be shared with the members of the VDS development team and major stakeholders to get feedback on the APIs as early as possible. The document is under SVN control
http://svn.hdfgroup.uiuc.edu/hdf5doc/trunk/RFCs/HDF5_Library/VirtualDataset/.

When checking the new version, update the name of the document and change page headers to reflect new version number and new date.

2 VDS APIs

2.1 VDS Creation

2.1.1 Modifications to H5Pset_layout and H5Pget_layout

We should update the documentation for `H5Pset(get)_layout` to add the new `H5D_VIRTUAL layout.`

2.1.2 H5Pset_virtual

Name: H5Pset_virtual
Signature:

herr_t H5Pset_virtual (*hid_t* dcpl_id, *hid_t* vspace_id, *const char* * src_file_name, *const char* * src_dset_name, *hid_t* src_space_id)

Purpose:

Sets the mapping between the virtual and source datasets.

Description:

H5Pset_virtual maps elements of the virtual dataset described by the virtual dataspace identifier vspace_id to the elements of the source dataset described by the source dataset dataspace identifier src_space_id. The source dataset is identified by the name of the file where it is located, src_file_name, and the name of the dataset, src_dset_name.

Parameters:

hid_t dcpl_id	-	IN: The identifier of the dataset creation property list that will be used when creating the virtual dataset.
hid_t vspace_id	-	IN: The dataspace identifier with the selection within the virtual dataset applied, possibly an unlimited selection.
const char * src_file_name	-	IN: The name of the HDF5 file where the source dataset is located. The file might not exist yet. The name can be specified using a C-style printf statement as described below.
const char * src_dset_name	-	IN: The path to the HDF5 dataset in the file specified by src_file_name. The dataset might not exist yet. The dataset name can be specified using a C-style printf statement as described below.
hid_t src_space_id	-	IN: The source dataset's dataspace identifier with a selection applied, possibly an unlimited selection.

When a selection with unlimited dimensions is used for the source dataset, the selection in the virtual dataset must also be an unlimited selection with the same number of unlimited dimensions. If fixed-size selections are used, the number of elements in the source dataset selection must be the same as the number of elements in the virtual dataset selection.

C-style printf Formatting Notes

C-style printf formatting allows a pattern to be specified in the name of a source file or dataset. Strings for the file and dataset names are treated as literals except for the following substitutions:

“%%”	-	Replaced with a single ‘%’ character.
“%*b*”	-	Where ‘b’ indicates that the block count of the selection in the unlimited dimension used. The full expression (for example, “%b”) is replaced with a single numeric value when the mapping is evaluated at VDS access time. Example code is available in the examples directory of the HDF5 distribution.

If the `printf` form is used for the source file or dataset names, the selection in the source dataset’s dataspace must be fixed-size; for more information see [1].

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

2.2 VDS Queries

2.2.1 H5Pget_virtual_count

Name: H5Pget_virtual_count
Signature:

herr_t H5Pget_virtual_count (*hid_t* dcpl_id, *size_t* *count)

Purpose:

Gets the number of mappings for the virtual dataset.

Description:

H5Pget_virtual_count gets the number of mappings for the virtual dataset that has a creation property list specified by the dcpl_id parameter.

Parameters:

hid_t dcpl_id	-	IN: The identifier of the virtual dataset creation property list.
size_t *count	-	IN: The number of mappings.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

2.2.2 H5Pget_virtual_vspace

Name: H5Pget_virtual_vspace
Signature:

hid_t H5Pget_virtual_vspace (*hid_t* dcpl_id, *size_t* index)

Purpose:

Gets a dataspace identifier for the selection within the virtual dataset used in the mapping.

Description:

H5Pget_virtual_vspace takes the dataset creation property list for the virtual dataset, dcpl_id, and the mapping index, index, and returns a dataspace identifier for the selection within the virtual dataset used in the mapping.

Parameters:

hid_t dcpl_id	-	IN: The identifier of the virtual dataset creation property list.
size_t index	-	IN: The mapping index. The index value i is $0 \leq i < M$, where M is number of mappings returned by the H5Pget_virtual_count function.

Returns:

Returns a dataspace identifier; otherwise returns a negative value.

2.2.3 H5Pget_virtual_srcspace

Name: H5Pget_virtual_srcspace
Signature:

hid_t H5Pget_virtual_srcspace (*hid_t* dcpl_id, *size_t* index)

Purpose:

Gets a dataspace identifier for the selection within the source dataset used in the mapping.

Description:

H5Pget_virtual_srcspace takes the dataset creation property list for the virtual dataset, dcpl_id, and the mapping index, index, and returns a dataspace identifier for the selection within the source dataset used in the mapping.

Parameters:

hid_t dcpl_id	-	IN: The identifier of the virtual dataset creation property list.
size_t index	-	IN: The mapping index. The index value i is $0 \leq i < M$, where M is number of mappings returned by the H5Pget_virtual_count function.

Returns:

Returns a dataspace identifier; otherwise returns a negative value.

2.2.4 H5Pget_virtual_filename

Name: H5Pget_virtual_filename
Signature:

ssize_t H5Pget_virtual_filename (*hid_t* dcpl_id, *size_t* index, *char* *name, *size_t* size)

Purpose:

Gets the filename of a source dataset used in the mapping.

Description:

H5Pget_virtual_filename takes the dataset creation property list for the virtual dataset, dcpl_id, and the mapping index, index, and retrieves a name of a file for a source dataset used in the mapping.

Up to size characters of the filename are returned in name; additional characters, if any, are not returned to the user application.

If the length of the filename, which determines the required value of size, is unknown, a preliminary call to H5Pget_virtual_filename with the last two parameters set to NULL can be made. The return value of this call will be the size in bytes of the filename. That value, plus 1 for a NULL terminator, is then assigned to size for a second H5Pget_virtual_filename call, which will retrieve the actual filename.

Parameters:

hid_t dcpl_id	-	IN: The identifier of the virtual dataset creation property list.
size_t index	-	IN: The mapping index. The index value i is 0≤i<M, where M is number of mappings returned by the H5Pget_virtual_count function.
char * name	-	OUT: A buffer containing the name of the file with the source dataset.
size_t size	-	IN: The size of the name buffer; must be the size of the file name in bytes plus 1 for a NULL terminator.

Returns:

Returns the length of the name if successful, otherwise returns a negative value.

2.2.5 H5Pget_virtual_dsetname

Name: H5Pget_virtual_dsetname
Signature:

ssize_t H5Pget_virtual_dsetname (*hid_t* dcpl_id, *size_t* index, *char* *name, *size_t* size)

Purpose:

Gets the name of a source dataset used in the mapping.

Description:

H5Pget_virtual_dsetname takes the dataset creation property list for the virtual dataset, dcpl_id, and the mapping index, index, and retrieves the name of a source dataset used in the mapping.

Up to size characters of the name are returned in name; additional characters, if any, are not returned to the user application.

If the length of the filename, which determines the required value of size, is unknown, a preliminary call to H5Pget_virtual_dsetname with the last two parameters set to NULL can be made. The return value of this call will be the size in bytes of the filename. That value, plus 1 for a NULL terminator, is then assigned to size for a second H5Pget_virtual_dsetname call, which will retrieve the actual filename.

Parameters:

hid_t dcpl_id	-	IN: The identifier of the virtual dataset creation property list.
size_t index	-	IN: The mapping index. The index value i is $0 \leq i < M$, where M is number of mappings returned by the H5Pget_virtual_count function.
char * name	-	OUT: A buffer containing the name of a source dataset.
size_t size	-	IN: The size of the name buffer; must be the size of the name in bytes plus 1 for a NULL terminator.

Returns:

Returns the length of the name if successful, otherwise returns a negative value.

2.3 VDS Access Properties

2.3.1 H5Pset_virtual_view

Name: H5Pset_virtual_view
Signature:

herr_t H5Pset_virtual_view (*hid_t* dapl_id, *H5D_vds_view_t* view)

Purpose:

Sets the view of VDS to include or exclude missing mapped elements.

Description:

H5Pset_virtual_view takes the access property list for the virtual dataset, dapl_id, and the flag, view, and sets the VDS view according to the flag value. The view will include all data before the first missing mapped data found if the flag is set to H5D_VDS_FIRST_MISSING or to include all available mapped data if the flag is set to H5D_VDS_LAST_AVAIALBLE. Missing mapped data will be filled with the fill value according to the VDS creation property settings. For VDS with unlimited mappings, the view defines the extent.

Parameters:

hid_t dapl_id	-	IN: The identifier of the virtual dataset access property list.
H5D_vds_view_t view	-	IN: The flag. Possible values are H5D_VDS_FIRST_MISSING or H5D_VDS_LAST_AVAIALBLE.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

2.3.2 H5Pget_virtual_view

Name: H5Pget_virtual_view
Signature:

herr_t H5Pget_virtual_view (*hid_t* dapl_id, *H5D_vds_view_t* *view)

Purpose:

Gets the view of VDS set for dataset access property list.

Description:

H5Pget_virtual_view takes the access property list for the virtual dataset, dapl_id, and gets the flag, view, set by the H5Pset_virtual_view call. The possible values of view are H5D_VDS_FIRST_MISSING or H5D_VDS_LAST_AVAIALBLE.

Parameters:

hid_t dapl_id	-	IN: The identifier of the virtual dataset access property list.
H5D_vds_view_t *view	-	OUT: The flag. Possible values are H5D_VDS_FIRST_MISSING or H5D_VDS_LAST_AVAIALBLE.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

2.3.3 H5Pset_virtual_printf_gap

Name: H5Pset_virtual_printf_gap
Signature:

herr_t H5Pset_virtual_printf_gap (*hid_t* dapl_id, *hsize_t* gap_size)

Purpose:

Sets the maximum number of missing source files and/or datasets with the printf-style names when getting the extent for unlimited VDS.

Description:

H5Pset_virtual_printf_gap sets the access property list for the virtual dataset, dapl_id, to instruct the library to stop looking for the mapped data stored in the files and/or datasets with the printf-style names after not finding gap_size files and/or datasets. The found source files and datasets will determine the extent of the unlimited VDS with the printf-style mappings.

For example, if regularly spaced blocks of VDS are mapped to datasets with the names d-1, d-2, d-3, …, d-N, …, and d-2 dataset is missing and gap_size is set to 0, then VDS will contain only data found in d-1. If d-2 and d-3 are missing and gap_size is set to 2, then VDS will contain the data from d-1, d-3, …, d-N,…. The blocks that are mapped to d-2 and d-3 will be filled according to the VDS fill value setting.

Parameters:

hid_t dapl_id	-	IN: The identifier of the virtual dataset access property list.
hsize_t gap_size	-	IN: The maximum number of the files and/or datasets allowed to be missing for determining the extent of the unlimited VDS with the printf style mappings.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

2.3.4 H5Pget_virtual_printf_gap

Name: H5Pget_virtual_printf_gap
Signature:

herr_t H5Pget_virtual_printf_gap (*hid_t* dapl_id, *hsize_t* *gap_size)

Purpose:

Gets the maximum number of missing source files and/or datasets with the printf-style names when getting the extent for unlimited VDS.

Description:

H5Pget_virtual_printf_gap gets the maximum number of missing printf-style files and/or datasets for determining the extent of the unlimited VDS, gap_size, using the access property list for the virtual dataset, dapl_id. The default library value for gap_size is 0.

Parameters:

hid_t dapl_id	-	IN: The identifier of the virtual dataset access property list.
hsize_t *gap_size	-	OUT: The maximum number of the files and/or datasets allowed to be missing for determining the extent of the unlimited VDS with the printf style mappings.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

2.4 Example 1: VDS creation

This example shows how the H5Pget_virtual_* APIS above are used. The source code of this example is available form http://svn.hdfgroup.uiuc.edu/hdf5/features/vds/examples/h5_vds.c

The program creates a virtual dataset that is displayed by h5dump as shown below:

```
$h5dump -pH vds.h5
HDF5 "vds.h5" {
GROUP "/" {
   DATASET "VDS" {
      DATATYPE  H5T_STD_I32LE
      DATASPACE  SIMPLE { ( 4, 6 ) / ( 4, 6 ) }
      STORAGE_LAYOUT {
         VIRTUAL {
             HYPERSLAB { (0,0)-(0,5) };
             a.h5;
             A;
             ALL ;
             HYPERSLAB { (1,0)-(1,5) };
             b.h5;
             B;
             ALL ;
             HYPERSLAB { (2,0)-(2,5) };
             c.h5;
             C;
             ALL;
            }
         }
      }
      FILTERS {
         NONE
      }
      FILLVALUE {
         FILL_TIME H5D_FILL_TIME_IFSET
         VALUE  -1
      }
      ALLOCATION_TIME {
         H5D_ALLOC_TIME_INCR
      }
   }
}
}
```

```
/************************************************************

  This example illustrates the concept of virtual dataset.
  The program creates three 1-dim source datasets and writes
  data to them. Then it creates a 2-dim virtual dataset and
  maps the first three rows of the virtual dataset to the data
  in the source datasets. Elements of a row are mapped to all
  elements of the corresponding source dataset.
  The fourth row is not mapped and will be filled with the fill
  values when virtual dataset is read back.

  The program closes all datasets, and then reopens the virtual
  dataset, and finds and prints its creation properties.
  Then it reads the values.

  This file is intended for use with HDF5 Library version 1.10

 ************************************************************/

#include "hdf5.h"
#include <stdio.h>
#include <stdlib.h>

#define FILE         "vds.h5"
#define DATASET      "VDS"
#define VDSDIM1         6
#define VDSDIM0         4
#define DIM0            6
#define RANK1           1
#define RANK2           2

const char *SRC_FILE[] = {
    "a.h5",
    "b.h5",
    "c.h5"
};

const char *SRC_DATASET[] = {
    "A",
    "B",
    "C"
};

int
main (void)
{
    hid_t        file, space, src_space, vspace, dset; /* Handles */
    hid_t        dcpl;
    herr_t       status;
    hsize_t      vdsdims[2] = {VDSDIM0, VDSDIM1},      /* Virtual satasets dimension */
                 dims[1] = {DIM0},           /* Source datasets dimensions */
                 start[2],                   /* Hyperslab parameters */
                 stride[2],
                 count[2],
                 block[2];
    int          wdata[DIM0],                /* Write buffer for source dataset */
```

```
                rdata[VDSDIM0][VDSDIM1],    /* Read buffer for virtual dataset */
                i, j, k, l;
    int         fill_value = -1;            /* Fill value for VDS */
    H5D_layout_t layout;                    /* Storage layout */
    size_t      num_map;                    /* Number of mappings */
    ssize_t     len;                        /* Length of the string; also a return value
*/
    char        *filename;
    char        *dsetname;
    hsize_t     nblocks;
    hsize_t     *buf;                       /* Buffer to hold hyperslab coordinates */

    /*
     * Create source files and datasets. This step is optional.
     */
    for (i=0; i < 3; i++) {
        /*
         * Initialize data for i-th source dataset.
         */
        for (j = 0; j < DIM0; j++) wdata[j] = i+1;

        /*
         * Create the source files and  datasets. Write data to each dataset and
         * close all resources.
         */

        file = H5Fcreate (SRC_FILE[i], H5F_ACC_TRUNC, H5P_DEFAULT, H5P_DEFAULT);
        space = H5Screate_simple (RANK1, dims, NULL);
        dset = H5Dcreate (file, SRC_DATASET[i], H5T_NATIVE_INT, space, H5P_DEFAULT,
                    H5P_DEFAULT, H5P_DEFAULT);
        status = H5Dwrite (dset, H5T_NATIVE_INT, H5S_ALL, H5S_ALL, H5P_DEFAULT,
                    wdata);
        status = H5Sclose (space);
        status = H5Dclose (dset);
        status = H5Fclose (file);
    }

    /* Create file in which virtual dataset will be stored. */
    file = H5Fcreate (FILE, H5F_ACC_TRUNC, H5P_DEFAULT, H5P_DEFAULT);

    /* Create VDS dataspace.  */
    space = H5Screate_simple (RANK2, vdsdims, NULL);

    /* Set VDS creation property. */
    dcpl = H5Pcreate (H5P_DATASET_CREATE);
    status = H5Pset_fill_value (dcpl, H5T_NATIVE_INT, &fill_value);

    /* Initialize hyperslab values. */
    start[0] = 0;
    start[1] = 0;
    count[0] = 1;
    count[1] = 1;
    block[0] = 1;
    block[1] = VDSDIM1;

    /*
```

```
     * Build the mappings.
     * Selections in the source datasets are H5S_ALL.
     * In the virtual dataset we select the first, the second and the third rows
     * and map each row to the data in the corresponding source dataset.
     */
    src_space = H5Screate_simple (RANK1, dims, NULL);
    for (i = 0; i < 3; i++) {
        start[0] = (hsize_t)i;
        /* Select i-th row in the virtual dataset; selection in the source datasets is the
same. */
        status = H5Sselect_hyperslab (space, H5S_SELECT_SET, start, NULL, count, block);
        status = H5Pset_virtual (dcpl, space, SRC_FILE[i], SRC_DATASET[i], src_space);
    }

    /* Create a virtual dataset. */
    dset = H5Dcreate (file, DATASET, H5T_NATIVE_INT, space, H5P_DEFAULT,
                dcpl, H5P_DEFAULT);
    status = H5Sclose (space);
    status = H5Sclose (src_space);
    status = H5Dclose (dset);
    status = H5Fclose (file);

    /*
     * Now we begin the read section of this example.
     */

    /*
     * Open the file and virtual dataset.
     */
    file = H5Fopen (FILE, H5F_ACC_RDONLY, H5P_DEFAULT);
    dset = H5Dopen (file, DATASET, H5P_DEFAULT);

    /*
     * Get creation property list and mapping properties.
     */
    dcpl = H5Dget_create_plist (dset);

    /*
     * Get storage layout.
     */
    layout = H5Pget_layout (dcpl);
    if (H5D_VIRTUAL == layout)
        printf(" Dataset has a virtual layout \n");
    else
        printf("Wrong layout found \n");

     /*
      * Find the number of mappings.
      */
     status = H5Pget_virtual_count (dcpl, &num_map);
     printf("Number of mappings is %d\n", num_map);

     /*
      * Get mapping parameters for each mapping.
      */
    for (i = 0; i < (int)num_map; i++) {
```

```
        printf("Mapping %d \n", i);
        printf("         Selection in the virtual dataset ");
        /* Get selection in the virttual  dataset */
        vspace = H5Pget_virtual_vspace (dcpl, (size_t)i);

        /* Make sure that this is a hyperslab selection and then print information. */
        if (H5Sget_select_type(vspace) == H5S_SEL_HYPERSLABS) {
            nblocks = H5Sget_select_hyper_nblocks (vspace);
            buf = (hsize_t *)malloc(sizeof(hsize_t)*2*RANK2*nblocks);
            status = H5Sget_select_hyper_blocklist (vspace, (hsize_t)0, nblocks, buf);
            for (l=0; l<nblocks; l++) {
                printf("(");
                for (k=0; k<RANK2-1; k++)
                    printf("%d,", (int)buf[k]);
                printf("%d ) - (", (int)buf[k]);
                for (k=0; k<RANK2-1; k++)
                    printf("%d,", (int)buf[RANK2+k]);
                printf("%d)\n", (int)buf[RANK2+k]);
            }
        }
        /* Get source file name. */
        len = H5Pget_virtual_filename (dcpl, (size_t)i, NULL, 0);
        filename = (char *)malloc((size_t)len*sizeof(char)+1);
        H5Pget_virtual_filename (dcpl, (size_t)i, filename, len+1);
        printf("         Source filename %s\n", filename);

        /* Get source dataset name. */
        len = H5Pget_virtual_dsetname (dcpl, (size_t)i, NULL, 0);
        dsetname = (char *)malloc((size_t)len*sizeof(char)+1);
        H5Pget_virtual_dsetname (dcpl, (size_t)i, dsetname, len+1);
        printf("         Source dataset name %s\n", dsetname);

        /* Get selection in the source dataset. */
        printf("         Selection in the source dataset ");
        src_space = H5Pget_virtual_srcspace (dcpl, (size_t)i);

        /* Make sure it is ALL selection and then print the coordinates. */
        if(H5Sget_select_type(src_space) == H5S_SEL_ALL) {
                printf("(0) - (%d) \n", DIM0-1);
        }
        H5Sclose(vspace);
        H5Sclose(src_space);
        free(filename);
        free(dsetname);
        free(buf);
    }

#ifdef LATER
    /*
     * Read the data using the default properties.
     */
    status = H5Dread (dset, H5T_NATIVE_INT, H5S_ALL, H5S_ALL, H5P_DEFAULT,
                rdata[0]);

    /*
     * Output the data to the screen.
```

```
     */
    printf (" VDS Data:\n");
    for (i=0; i<VDSDIM0; i++) {
        printf (" [");
        for (j=0; j<VDSDIM1; j++)
            printf (" %3d", rdata[i][j]);
        printf ("]\n");
    }
#endif
    /*
     * Close and release resources.
     */
    status = H5Pclose (dcpl);
    status = H5Dclose (dset);
    status = H5Fclose (file);

    return 0;
}
```

3 New APIs in support of VDS feature

3.1 H5S APIs

This section contains new APIs for the H5S interface needed to query properties of the regular hyperslab selections.

3.1.1 H5Sis_regular_hyperslab

Name: H5Sis_regular_hyperslab
Signature:

htri_t H5Sis_regular_hyperslab (*hid_t* space_id)

Purpose:

Determines if a hyperslab selection is regular.

Description:

Regular hyperslab selection is a hyperslab selection described by setting the offset, stride, count and block parameters to the H5Sselect_hyperslab call. If several calls to H5Sselect_hyperslab are needed, then the hyperslab selection is irregular.

H5Sis_regular_hyperslab takes the dataspace identifier, space_id, and queries the type of the hyperslab selection.

Parameters:

hid_t space_id - IN: The identifier of the dataspace.

Returns:

Returns TRUE/FALSE for hyperslab selection; FAIL on error or when querying other selection types such as point selection.

3.1.2 H5Sget_regular_hyperslab

Name: H5Sget_regular_hyperslab
Signature:
herr_t H5Sget_regular_hyperslab (*hid_t* space_id, *hsize_t* start[], *hsize_t* stride[], *hsize_t* count[], *hsize_t* block[])

Purpose:
Retrieves a regular hyperslab selection.

Description:
Regular hyperslab selection is a hyperslab selection described by setting the offset, stride, count and block parameters to the H5Sselect_hyperslab call. If several calls to H5Sselect_hyperslab are needed, then the hyperslab selection is irregular.

H5Pget_regular_hyperslab takes the dataspace identifier, space_id, and retrieves the values of start, stride, count, and block for the regular hyperslab selection.

Note:
Note that if a hyperslab selection is originally regular, then becomes irregular through selection operations, and then becomes regular again, the new final regular selection may be equivalent but not identical to the original regular selection.

Parameters:

hid_t space_id	-	IN: The identifier of the dataspace.
hsize_t start[]	-	OUT: Offset of start of a regular hyperslab.
hsize_t stride[]	-	OUT: Stride of the regular hyperslab.
hsize_t count[]	-	OUT: Number of blocks in the regular hyperslab.
hsize_t block[]	-	OUT: Size of block in the regular hyperslab.

Returns:
Returns the non-negative value if successful, otherwise returns a negative value.

References:

1. "RFC: HDF5 Virtual Dataset", The HDF Group, https://confluence.hdfgroup.uiuc.edu/display/HDFExternal/HDF5+Virtual+Dataset

Revision History

February 10, 2015	Version 1 circulated for comment within The HDF Group.
February 11, 2015	Version 2. Names of the APIs were updated according to suggestions made during the code review session on 2/10/2015. Document circulated for comment within The HDF Group.
February 24, 2015	Version 3. Added an example for the H5Pget_virtual_* functions.
March 2, 2015	Version 4. Added section for the H5S* functions.