with 10 rows then you would have a 10-row dataset, each element of which consists of one 5 x 4 array of signed longs.

In Igor this could be treated as a 3D wave consisting of 5 rows, 4 columns and 10 layers. However, since Igor supports only 4 dimensions while HDF5 supports 32, you could easily run out of dimensions in Igor. Therefore when you load an H5T_ARRAY-type dataset, HDF5LoadData creates a wave whose dimensionality is that of the array type, not that of the underlying dataset, except that the highest dimension is increased to make room for all of the data. This reduces the likelihood of running out of dimensions.

HDF5LoadData can not load array data whose base type is compound or array (see **HDF5 Nested Datatypes** on page II-211 for details).

The following example illustrate how to load an array datatype. The example uses a sample file provided by NCSA and stored in Igor's HDF5 Samples directory.

Create an Igor symbolic path (Misc→New Symbolic Path) named HDF5Samples which points to the folder containing the SDS_array_type.h5 file (Igor Pro X Folder:Examples:Feature Demos:HDF5 Samples).

The sample file contains a 10 row 1D data set, each element of which is a 5 row by 4 column matrix of 32-bit signed big-endian integers (a.k.a., big-endian signed longs or I32BE).

When we load the dataset, we get 5 rows and 40 columns.

```
Variable fileID
HDF5OpenFile/P=HDF5Samples /R fileID as "SDS_array_type.h5"
HDF5LoadData fileID, "IntArray"
HDF5CloseFile fileID
Edit IntArray
```

The first four columns contain the 5x4 matrix from row zero of the dataset. The next four columns contain the 5x4 matrix from row one of the dataset. And so on.

In this case, Igor has enough dimensions so that you could, if you want, reorganize it as a 3D wave consisting of 10 layers of 5x4 matrices. You would do that using this command:

```
Redimension /N=(5,4,10) /E=1 IntArray
```

The /E=1 flag tells Redimension to change the dimensionality of the wave without actually changing the stored data. In Igor, the layout in memory of a 5x4x10 wave is the same as the layout of a 5x40 wave. The redimension merely changes the way we look at the wave from 40 columns of 5 rows to 10 layers of 4 columns of 5 rows.

Although you can load a dataset with an array datatype, Igor currently provides no way to write a datatype with an array datatype.

Loading HDF5 Reference Data

Most HDF5 files do not use reference datatypes so most users do not need to know this information.

An HDF5 dataset or attribute can contain references to other datasets, groups and named datatypes. There are two types of references: "object references" and "dataset region references". **HDF5LoadData** loads both types of references into text waves.

An element of an object reference dataset can refer to a dataset in the same or in another file. An element of a dataset region reference dataset can refer only to a dataset in the same file.

Loading HDF5 Object Reference Data

For each object reference to a dataset, HDF5LoadData returns "D:" plus the full path of the dataset within the HDF5 file, for example, "D:/GroupA/DatasetB".

For references to groups and named datatypes, HDFLoadData returns "G:" and "T:" respectively, followed by the path to the object.

Loading HDF5 Dataset Region Reference Data

This section is for advanced HDF5 users. A demo experiment, "HDF5 Dataset Region Demo.pxp", provides examples and utilities for dealing with dataset region references.

Prior to Igor Pro 9.00, HDF5LoadData returned the same thing when loading an object reference or a dataset region reference. It had this form:

```
<object type character>:<full path>
```

For a dataset, this might be something like

```
D:/Group1/Dataset3
```

In Igor Pro 9.00 and later, HDF5LoadData returns additional information for a dataset region reference. It has this form with the additional information shown in red:

```
<object type character>:<full path>.<region info>
```

For a dataset, this might be something like this with the additional information shown in red:

```
D:/Group1/Dataset3.REGIONTYPE=BLOCK;NUMDIMS=2;NUMELEMENTS=2;COORDS=0,0-
0,2/0,11-0,13;
```

If you have code that depends on the pre-Igor9 behavior, you can make it work with Igor9 by using StringFromList with "." as list separator string to obtain the text preceding the dot character.

The region info string is a semicolon-separated keyword-value string constructed for ease of programmatic parsing. It consists of these parts:

```
REGIONTYPE=<type>
NUMDIMS=<number of dimensions>
NUMELEMENTS=<number of elements>
COORDS=<list of points> or <list of blocks>
```

<type> is POINT for a region defined as a set of points, BLOCK for a region defined as a set of blocks.

<number of dimensions> is the number of dimensions in the dataset.

<number of elements> is the number of points in a region defined as a set of points or the number of blocks in a region defined as a set of blocks.

<list of points> has the following format:

```
<point coordinates>/<point coordinates>/...
```

where <point coordinates> is a comma-separated list of coordinates.

For a 2D dataset with three selected points, this might look like this:

```
3,4/11,13/21,30
```

which specifies these three points:

```
row 3, column 4
row 11, column 13
row 21, column 30
```

<list of blocks> has the following format:

```
<coordinates>-<coordinates>/<coordinates>-<coordinates>/...
```

where <coordinates> is a comma-separated list of coordinates.