As of this writing, the section 4.2.3, "HDF5 Path Names" of the HDF5 User's Guide says:

Component link names may be any string of ASCII characters not containing a slash or a dot ("/"and ".", which are reserved as noted above). However, users are advised to avoid the use of punctuation and non-printing characters, as they may create problems for other software.

HDF5 Data Types Versus Igor Data Types

HDF5 supports many data types that Igor does not support. When loading such data types, Igor attempts to convert to a data type that it supports, if possible. In the process, precision may be lost.

By default and for backward compatibility, Igor saves and loads HDF5 64-bit integer data as double-precision floating point. Precision may be lost in this conversion. To save and load 64-bit integer data as 64-bit integer, use /OPTS=1 with **HDF5SaveData**, **HDF5SaveGroup**, **HDF5LoadData** and **HDF5LoadGroup**. Because most operations are carried out in Igor in double-precision floating point, we recommend loading 64-bit integer as double if it fits in 53 bits and as 64-bit integer if it may exceed 53 bits.

Since Igor does not currently support 128-bit floating point data (long double), Igor loads HDF5 128-bit floating point data as double-precision floating point. Precision is lost in this conversion.

HDF5 Max Dimensions Versus Igor Max Dimensions

The HDF5 library supports data with up to 32 dimensions. Igor supports only four dimensions. If you load HDF5 data whose dimensionality is greater than four, the **HDF5LoadData** operation creates a 4D wave with enough chunks to hold all of the data. ("Chunk" is the name of the fourth dimension in Igor, after "row", "column" and "layer".)

For example, if you load a dataset whose dimensions are 7x6x5x4x3x2, HDF5LoadData creates a wave with dimensions 7x6x5x24. The wave has 24 chunks. The number 24 comes from multiplying the number of chunks in the HDF5 dataset (4 in this case) by the size of each higher dimension (3 and 2 in this case).

Row-major Versus Column-major Data Order

HDF5 and Igor store multi-dimensional data differently in memory. For almost all purposes, this difference is immaterial. For those rare cases in which it matters, here is an explanation.

HDF5 stores multi-dimensional data in "row-major" order. This means that the index associated with the highest dimension changes fastest as you move in memory from one element to the next. For example, if you have a two-dimensional dataset with 5 rows and 4 columns, as you move sequentially through memory, the column index changes fastest and the row index slowest.

Igor stores data in "column-major" order. This means that the index associated with the lowest dimension changes fastest as you move in memory from one element to the next. For example, if you have a two-dimensional dataset with 5 rows and 4 columns, as you move sequentially through memory, the row index changes fastest and the column index slowest.

To work around this difference, after the **HDF5LoadData** or **HDF5LoadGroup** operation initially loads data with two or more dimensions, it shuffles the data around in memory. The result is that the data when viewed in an Igor table looks just as it would if viewed in the HDF Group's HDFView program although its layout in memory is different. A similar accomodation occurs when you save Igor data using **HDF5SaveData** or **HDF5SaveGroup**.

HDF5 Images Versus Igor Images

The HDF5 Image and Palette Specification provides detailed guidelines for storing an image and its associated information (e.g., palette, color model) in an HDF5 file. However many HDF5 users do not follow the specification and just write image data to a regular 2D dataset. To distinguish between these two ways of storing an image, we use the term "formal image" to refer to an image written to the specification and "regular image" to refer to regular 2D datasets which the user thinks of as an image.