Loading worksheets on demand

This feature, new in version 0.7.1, is governed by the `on_demand` argument to the `open_workbook()` function and allows saving memory and time by loading only those sheets that the caller is interested in, and releasing sheets when no longer required.

`on_demand=False` **(default):**

No change. `open_workbook()` loads global data and all sheets, releases resources no longer required (principally the `str` or `mmap.mmap` object containing the Workbook stream), and returns.

`on_demand=True` **and BIFF version < 5.0:**

A warning message is emitted, `on_demand` is recorded as `False`, and the old process is followed.

`on_demand=True` **and BIFF version >= 5.0:**

`open_workbook()` loads global data and returns without releasing resources. At this stage, the only information available about sheets is `Book.nsheets` and `Book.sheet_names()`.

`Book.sheet_by_name()` and `Book.sheet_by_index()` will load the requested sheet if it is not already loaded.

`Book.sheets()` will load all unloaded sheets.

The caller may save memory by calling `Book.unload_sheet()` when finished with the sheet. This applies irrespective of the state of `on_demand`.

The caller may re-load an unloaded sheet by calling `Book.sheet_by_name()` or `Book.sheet_by_index()`, except if the required resources have been released (which will have happened automatically when `on_demand` is false). This is the only case where an exception will be raised.

The caller may query the state of a sheet using `Book.sheet_loaded()`.

`Book.release_resources()` may used to save memory and close any memory-mapped file before proceeding to examine already-loaded sheets. Once resources are released, no further sheets can be loaded.

When using on-demand, it is advisable to ensure that `Book.release_resources()` is always called, even if an exception is raised in your own code; otherwise if the input file has been memory-mapped, the `mmap.mmap` object will not be closed and you will not be able to access the physical file until your Python process terminates. This can be done by calling `Book.release_resources()` explicitly in the finally part of a try/finally block.

The Book object is also a context manager, so you can wrap your code in a `with` statement that will make sure underlying resources are closed.