Paper 242-28

Programming Standards, Style Sheets, and Peer Reviews: A Practical Guide

Dianne Louise Rhodes, Westat, Rockville, MD

ABSTRACT

When a new job starts, or a new manager takes over a project, the programming team is faced with a new culture. Often, this means that they are asked to adhere to yet another set of programming standards, use style sheets, and are subject to peer reviews. This process is usually short lived, because of the lack of a practical approach to applying standards and enforcement. This paper goes through a step by step process of developing programming standards, classifying them and entering them into a database. This database can then be used to develop style sheets and check lists for peer review and testing. Through peer reviews and in preparation for them, programmers learn good programming practices.

INTRODUCTION

In researching this paper, I found a number of sources of programming standards, some specific to SAS and others more general (see References). Everyone seems to agree that standards are a good thing, but rarely did I find mention of a practical approach to the use and enforcement of standards. No one wants to police programmers; that has been compared to "herding cats." Given explicit conventions to follow, programmers can easily review each other's work and apply these standards. When the corporate culture encourages peer reviews by allocating time to this activity it is an opportunity for professional growth. The process for developing explicit standards and checklists for peer reviews are detailed in this paper.

WHY HAVE STANDARDS AND STYLE SHEETS?

In a maintenance environment, a programmer should be able to make changes to a program without fully comprehending the entire process. If the original programmer has followed standards, which always include comments, it will be easier to understand the code (Aster, 1999).

The majority of standards fall into the category of documentation. These are rules that make the code easier to read on the page, easier to follow the logic and logical branches, and leave less room for interpretation. Code that meets these requirements is easier to maintain.

Standards are useful working in a teamwork development environment because they set minimum requirements, which in turn insure some uniformity from programmer to programmer. They are imperative in managing a large project, where source code control is also an issue. (See Whitney, 1999).

Standards for the sake of having standards are a good idea as well. Aster (1999) says: "A good style is simple, clear, and consistent... The main point, however, is that just about any style that you follow consistently is better than no style. Even a bizarrely idiosyncratic coding style can eventually start to make sense to the reader. But if your code has no style, a maintenance programmer can never quite figure it out without reading every detail."

If you are the only one who will be reading your code, you may be asking yourself, "Why bother?" As Fehd (2000) points out in his "Writing for Reading SAS Style Sheet", you are coding for two events – the machine at execution time, and yourself (or another programmer) when you revisit your code a year from now and try to remember the purpose and function of a specific program.

WHY HAVE PEER REVIEWS?

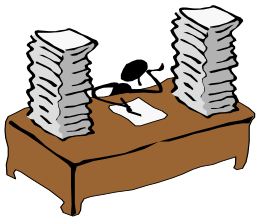
Peer reviews and formal walkthroughs are a valuable exercise in quality control. These processes help to ensure that code complies with programming standards, meets specifications, and is error-free. They are critical in managing a large project to ensure uniformity in programming style and use of variable names (Whitney, 1999).

Often this process breaks down as a project progresses and deadlines loom large. This is precisely the situation that benefits the most by having a second or third pair of eyes examining your work. A good Peer Review Process demands a corporate culture that adopts it and embraces it.

FACILITATING PEER REVIEWS AND WALKTHROUGHS

An informal walkthrough requires preparation time on the part of the reviewer, the programmer, and others critical to the review process. The "others" will vary as different corporate cultures may include managers, for example, while others specifically exclude them. Once time is allocated, a helpful tool is to have a checklist. This gives the reviewer an objective set of criterion to apply against the program or code being reviewed. The checklists will vary depending on the lifecycle stage of the programming, e.g., analysis, design, implementation, testing, maintenance.

THE DILEMMA



Despite the promises of a "paperless office" made in the '80's, I have accumulated a variety of standards, style sheets, testing procedures, hints and tips. Mostly these are a rag-tag bunch of paper documents, with something of value lurking in their depths. Some were developed for a particular project and are therefore very specific; others are generic "wish lists" of programmer behavior. The dilemma – how to translate all these piles of paper into "useable" standards?

RULES OF DECLUTTERING

I approached this problem using some of the tricks I have learning to "declutter" my living and working spaces. The **first rule**: get everything out in the open. The **second rule:** group similar items together. Eliminate duplicates. The **third rule**: find homes for everything. And the **fourth rule**: throw out or give away anything you haven't used in over a year. That might be harder to do with standards than with old clothes, but the idea is the same – if you aren't using it, it's probably obsolete.

APPLYING THE RULES TO STANDARDS

The **first task** was to locate copies of all the standards, rules, and tips I wanted to include. This meant I had to dig through files and piles of papers to find all the goodies I wanted to include in this project. I also printed out various SUGI and NESUG papers appropriate to the subject. The **second task**: put like objects together, i.e. Classify them. I initially defined four groups of rules: documentation, efficiency, maintenance, and testing (with some overlap). The **third task**: put them into a database. I elected to use MS Access® because it was quick and easy and I could give the data entry work to someone who wasn't familiar with SAS. SAS was used to develop more complex reports and checklists. I also tried to operationalize the rules, that is, give working examples. The **fourth task**: I eliminated some rules that were vague or subjective such as "avoid unnecessary branches."

CORPORATE STANDARDS

Does your company have corporate standards? When I asked this question of my colleagues, the answer was no. Did I want to be the one to try to create corporate standards? No. For this reason, I decided to keep most of my styles and tips pretty general, instead of implementing specific rules. For example, I suggest that it is useful to indent code to show logical flow. But I don't specify a style, since I don't really care what style is used as long as it is used consistently.

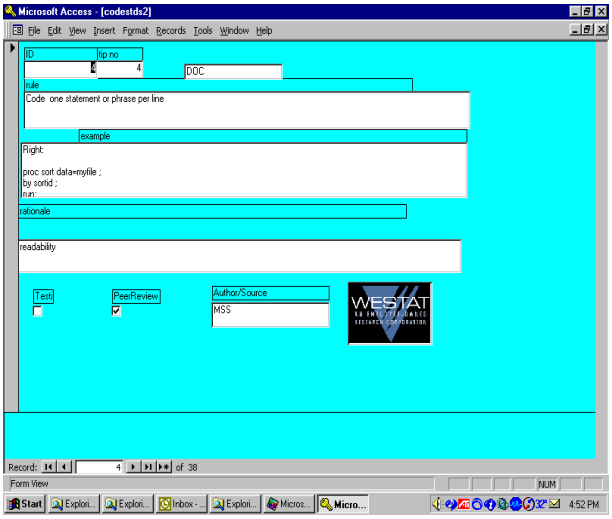
THE TIPS DATABASE

Working from various paper documents, I roughed out a database. The fields were tip number, type, the rule, an example, and the rationale for the rule. I quickly added check boxes for Peer Review and Testing, and a field to store the author or source. This allows for the generation of various checklists for peer reviews, testing procedures, etc. and customization by selecting certain subgroups. The Appendix contains examples of a document to present tips and standards, checklists used for peer reviews and testing, and a report that can be used as a style sheet.

The types that I defined were:
Documentation. Tips or standards that help document the purpose of the code and make it easier to read.
Efficiency. Tips to make the code more efficient; that is to run faster.
Maintenance. Tips to make the code easier to maintain. These sometimes are contrary to the efficiency tips!
Testing. These tips mainly concerned testing issues, or things to look for when testing. This is an area I would like to continue to enhance.

FORMS

The main form was a simple data entry form. Here the user transcribes from a paper cheat sheet the key elements of the tip, trick, or standard.



REPORTS

The first report is simply a hard copy of the database, as shown in Programming Tips and Tricks in the Appendix. These are listed by tip number, which is simply the order in which they were entered into the database. I included another tip number field in the database, so that I could change the order of the tips at a future date.

The Style Sheet is a list of the tips, primarily those related to documentation.

The Peer Review checklist was produced from a query that selected the tips where the Peer Review check box was checked. It is a quick list for programmers to review when they are desk checking a program.

The Testing checklist was produced from a query where the Test check box was checked. It is a list for review when testing a program, which could be stress testing, parallel testing before putting a program into production, etc.

ENHANCEMENTS

The major enhancement I plan to make is to create another table and input form to allow programmers to interact with the Peer Review and Test checklists while desk checking a program. This would allow them to check off that a program is in compliance or needs work in a particular area.

CONCLUSION

Organizing standards and style sheets can be an onerous task, but it's a necessary requisite to implementing their use. Putting them all into one place, cleaning them up, and having them where they are accessible to the programming staff is part of the process of learning good programming practices.

DISCLAIMER: The contents of this paper are the work of the author(s) and do not necessarily represent the opinions, recommendations, or practices of Westat.

REFERENCES

Aster, Rick (1998).“Coding for Posterity.” In the 11th Annual Proceedings of the NorthEast SAS Users Group. http://www.nesug.org/Proceedings/nesug98/appl/p131.pdf

Carpenter, Arthur L. (1999). “Getting More for Less: A Few SAS Programming Efficiency Issues.” In the 12th Annual Proceedings of the NorthEast SAS Users Group
http://www.nesug.org/Proceedings/nesug99/cc/CC199.PDF

Cheng, Alice M. (1999).“Robust Programming Techniques in the SAS System.” In the 12th Annual Proceeding of the NorthEast SAS Users Group.
http://www.nesug.org/Proceedings/nesug99/po/PO159.PDF

Fehd, Ronald J. (2000). “The Writing for Reading SAS Style Sheet: Tricks, Traps & Tips from SAS-L’s Macro Maven.” In the 25th Annual Proceedings of the SAS Users Group International.
http://www2.sas.com/proceedings/sugi25/25/ad/25p038.pdf

Whitney, C. Michael (1999) “Taming the Chaos: Managing Large SAS/AF Applications Using Programming Standards and the Source Control Manager of Version 7 of the SAS System.” In the 24th Proceedings of the SAS Users Group International.
http://www2.sas.com/proceedings/sugi24/AppDevel/p10-24.pdf

ACKNOWLEDGMENTS

I want to thank Duke Owen for giving me the opportunity to teach a class on programming best practices at Westat. This gave me the time to research and put together some of the ideas presented here. I extend my appreciation to The Macro Maven as a valuable resource and thanks for reading my first drafts and whispering SQL in my ear.

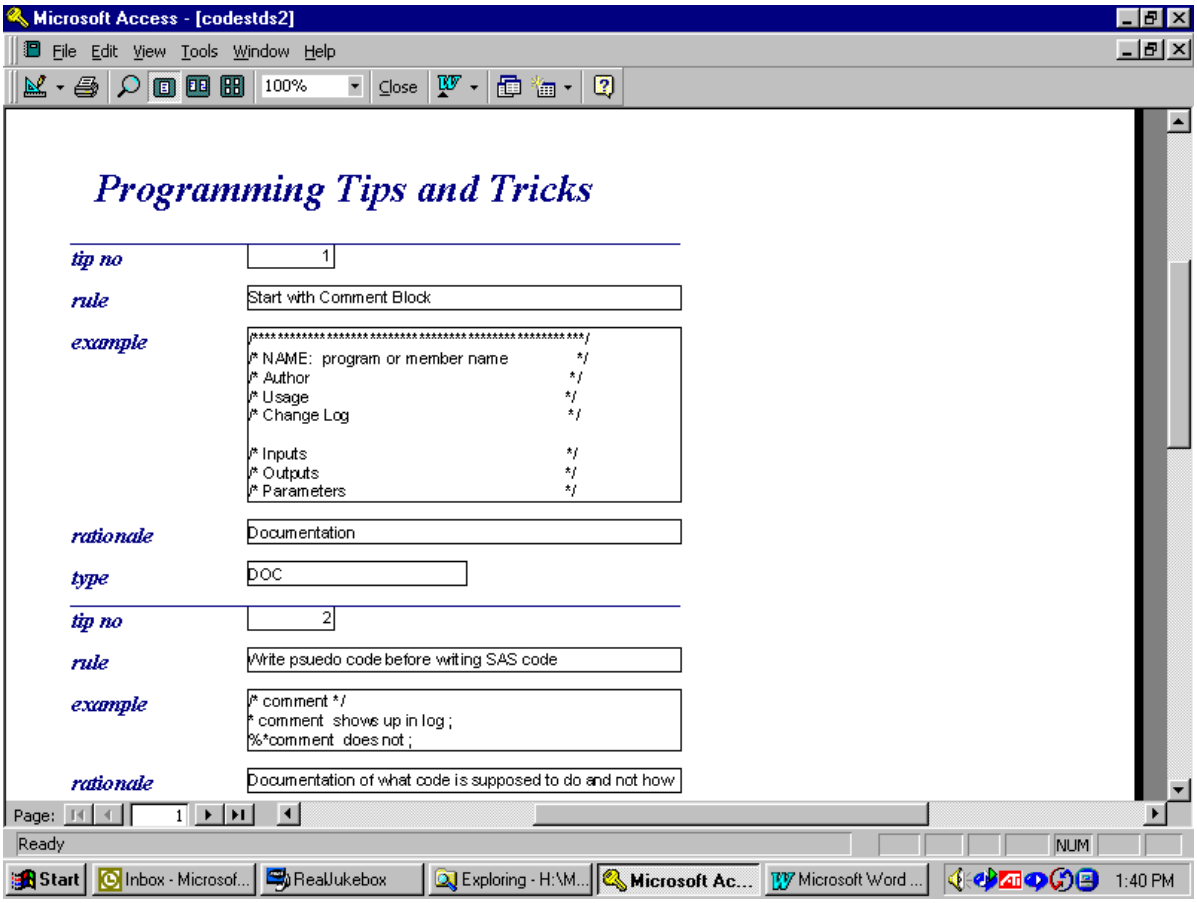
CONTACT INFORMATION

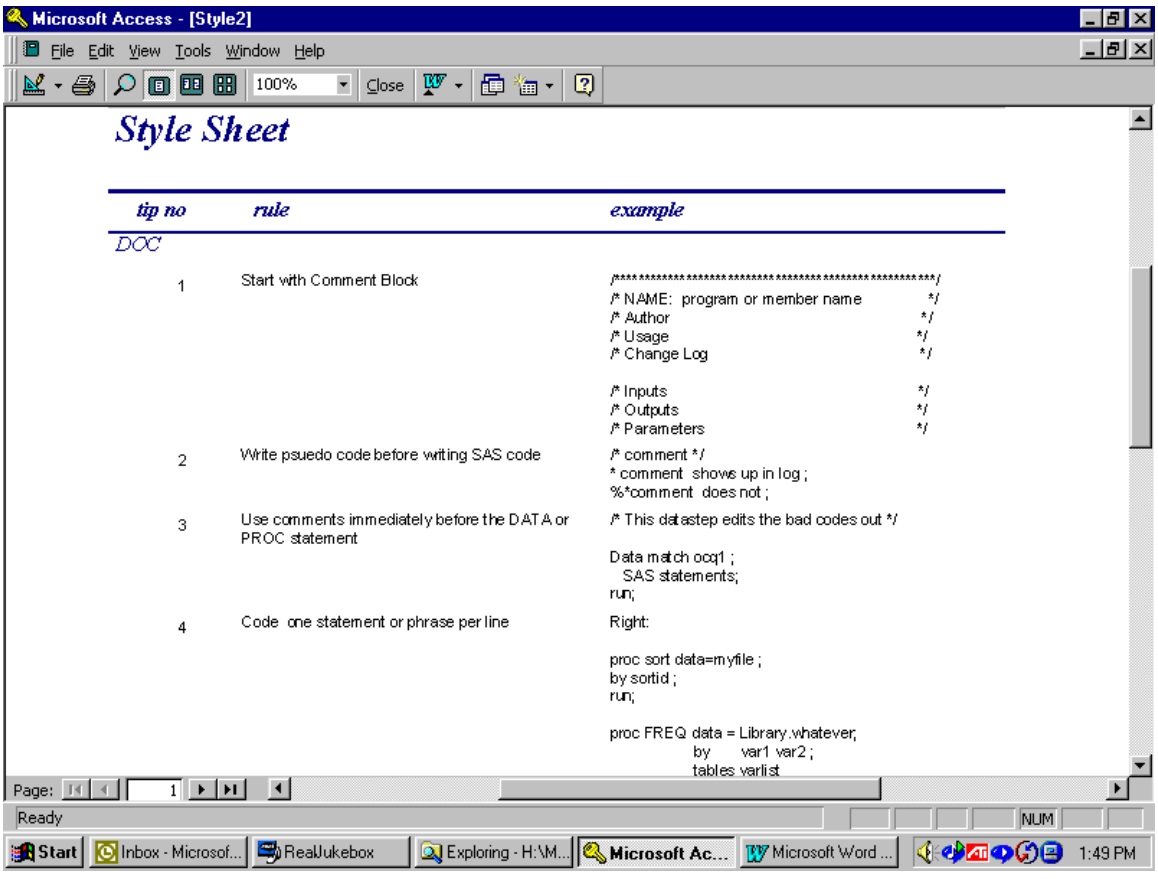
Comments, questions, and additions are welcomed.
Contact the author at:

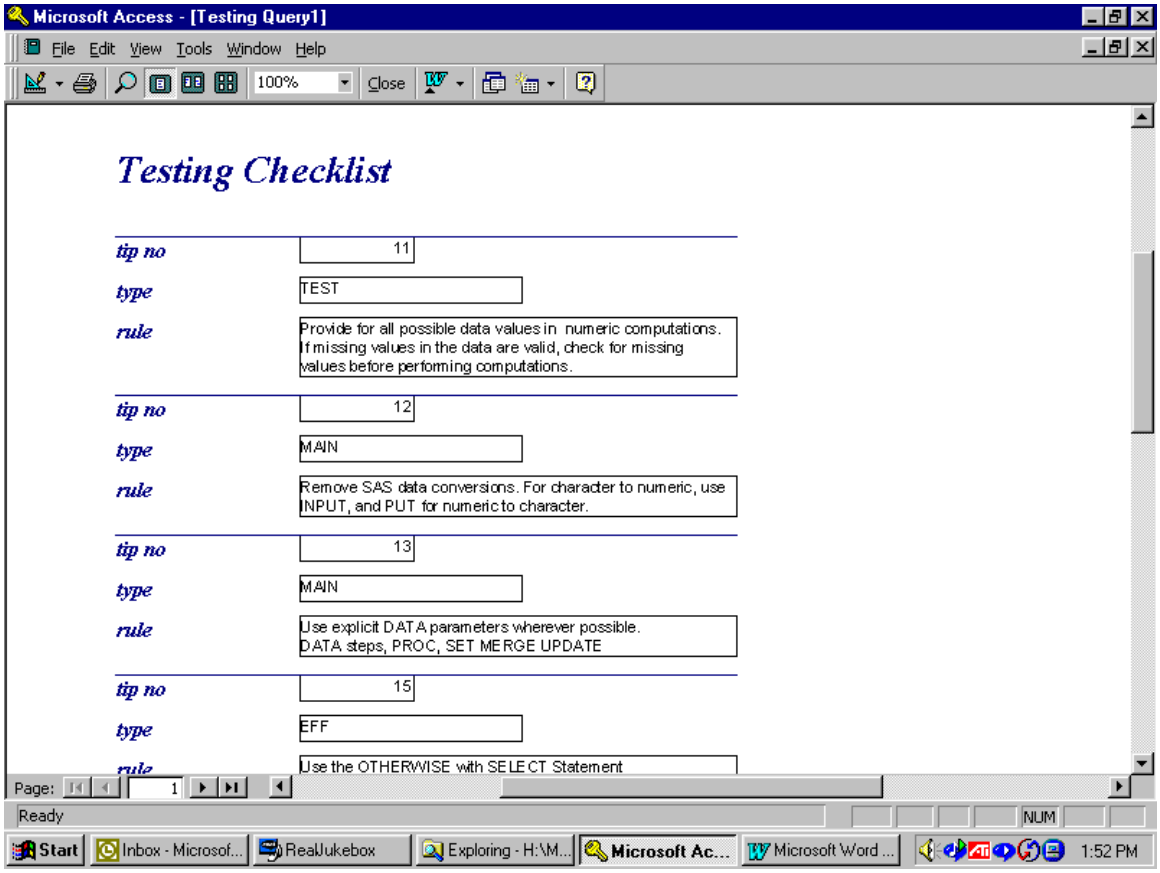
Dianne Louise Rhodes
WESTAT
An Employee-Owned Research Corporation
1650 Research Blvd.
Rockville, MD 20850
Phone: (301) 315-5977
Email: diannerhodes@westat.com

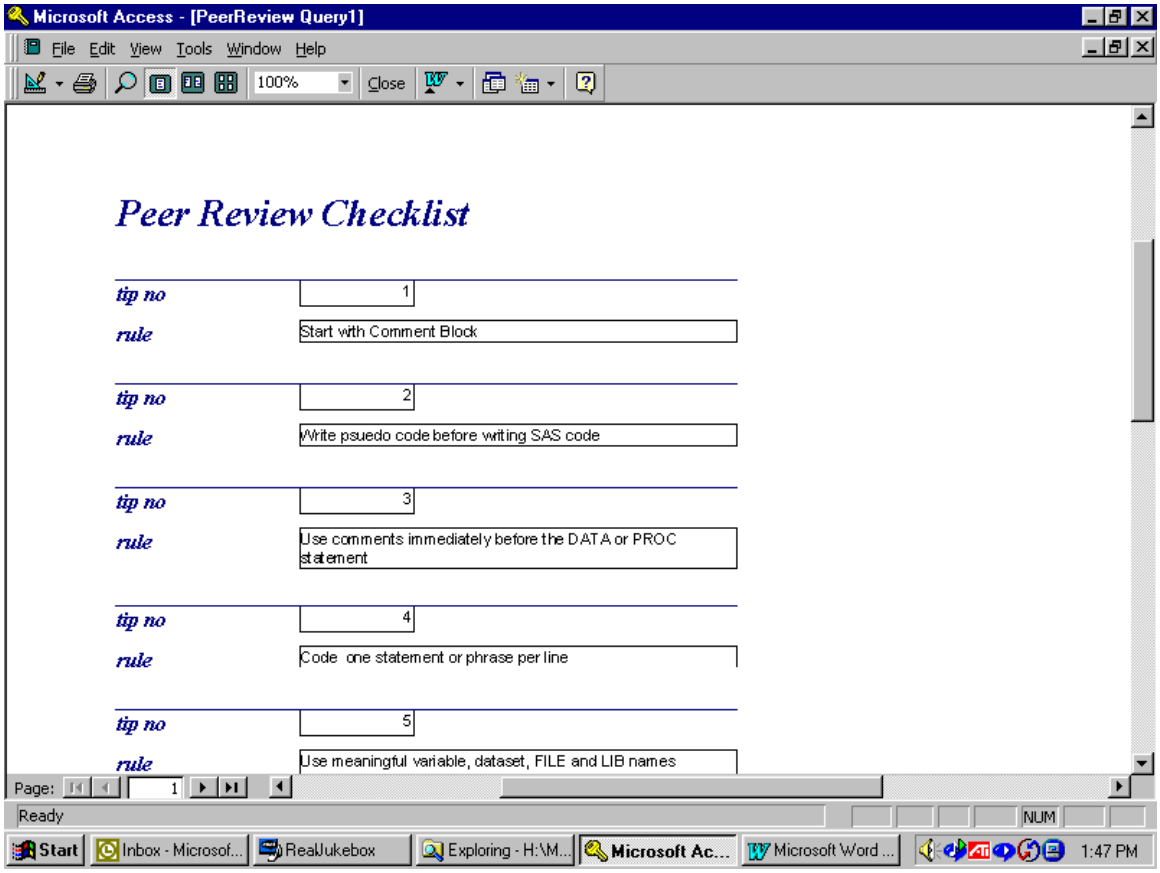
My intent was to include a copy of this MS Access® database with the SUGI 28 proceedings. However, if it is not available, contact me and I can provide it for you.

Appendix of Reports






Microsoft Access - [Testing Query1]
File Edit View Tools Window Help
100%
Close
Testing Checklist
tip no 11
type TEST
rule Provide for all possible data values in numeric computations. If missing values in the data are valid, check for missing values before performing computations.
tip no 12
type MAIN
rule Remove SAS data conversions. For character to numeric, use INPUT, and PUT for numeric to character.
tip no 13
type MAIN
rule Use explicit DATA parameters wherever possible. DATA steps, PROC, SET MERGE UPDATE
tip no 15
type EFF
rule Use the OTHERWISE with SELECT Statement
Page: 1
Ready
NUM
Start
Inbox - Microsof...
RealJukebox
Exploring - H:\M...
Microsoft Ac...
Microsoft Word ...
1:52 PM


Microsoft Access - [PeerReview Query1]
File Edit View Tools Window Help
100%
Close
Peer Review Checklist
tip no 1
rule Start with Comment Block
tip no 2
rule Write psuedo code before writing SAS code
tip no 3
rule Use comments immediately before the DATA or PROC statement
tip no 4
rule Code one statement or phrase per line
tip no 5
rule Use meaningful variable, dataset, FILE and LIB names
Page: 1
Ready
NUM
Start
Inbox - Microsof...
RealJukebox
Exploring - H:\M...
Microsoft Ac...
Microsoft Word ...
1:47 PM