PharmaSUG 2019 - Paper DV-297

Building Automations for Generating R and SAS Code Supporting Visualizations Across Multiple Therapeutic Areas

Anastasia Alexeeva, William Martersteck, and Mei Zhao, Eli Lilly and Co

ABSTRACT

Besides the typical portfolio work statistical programmers perform throughout the course of a clinical trial, there may be a need to create additional safety and efficacy visualizations across numerous trials. The motivation for these requests may be to understand clinical data better, or to use results of analyses to make important business decisions about a compound or indication. In order to do this, analysts write new programs that generate the outputs, create and validate input data sets, and author specifications for input data sets and outputs. Whether individual data sets or analyses are executed in SAS, R, or another programming language is subject to the preference of the programmer and the business need. This paper demonstrates an example of how to automate the generation of SAS and R programs by the use of a trial-specific file that contains global variable assignments. An R program reads this file and passes its contents to functions that create all the R, SAS, and Rmd files needed for the project. The global variables provide the key study specific information to populate the program headers, as well as everything needed to create and validate the data, and generate the output and the specifications for proper documentation. This strategy allows us to seamlessly leverage the benefits of R and SAS in one project; in addition, storing important programs in the Github hosting service and using the Git revision control system offer the opportunity for other teams to easily apply reference code to another study without having to make tedious modifications to many programs in order to generate results.

INTRODUCTION

In order to facilitate consistency among clinical trials and a uniform approach to analysis of data, pharmaceutical industries place great effort into standardization of programs, data sets, and outputs. By now, it is relatively simple to automate the production of a variety of static Tables, Figures, and Listings (TFLs) across different studies and disease indications. However, standardizing the production of a more complex project, such as a compilation of interactive visualizations is much more difficult. This paper presents a method that was implemented to automate and streamline the generation of code and data for interactive visualizations across multiple trials, describes the technologies employed, and summarizes the improvements in the process that resulted.

The motivation for these innovations came from a need to create interactive visualizations to help with the analysis and discussion of trial efficacy results. Trial programmers were requested to provide a series of interactive visualizations and animations (created in Spotfire and R) to assist in the exploration of the data in the few days immediately after database lock occurred.

For the first few iterations of this project, the implementation process contained a lot of manual steps: hand-generating requirements for data sets and outputs and tedious coding to create visualization data sets. Often this consisted of manually copying from existing code, which is error-prone, and resulted in a lot of work that needed to be redone for each new trial. This procedure required a lot of effort to validate the code and ensure that it meets the requirements, which does not guarantee identifying all errors, while adding to the workload. In order to eliminate these issues, we streamlined the process by automating both the creation of requirements and the data sets using trial-specific meta-data; this work resulted in the tool described in this paper. 1

The automation tool, stored in Github, generates SAS and R codes that generate the data sets, which are used to create interactive graphics and animations. This tool can be easily modified to adapt to different clinical trials using trial and visualization metadata, which stores common elements that could have different values across the trials.

Each trial has different goals, depending on the therapeutic area, indication, and primary endpoints. These heterogeneous goals drive different trial designs, different data semantics and/or different analyses. To standardize and automate processes within such a diverse environment requires flexible

tools that can readily accommodate to study-specific context and goals. The automation tool presented in this paper sought to satisfy the following objectives:

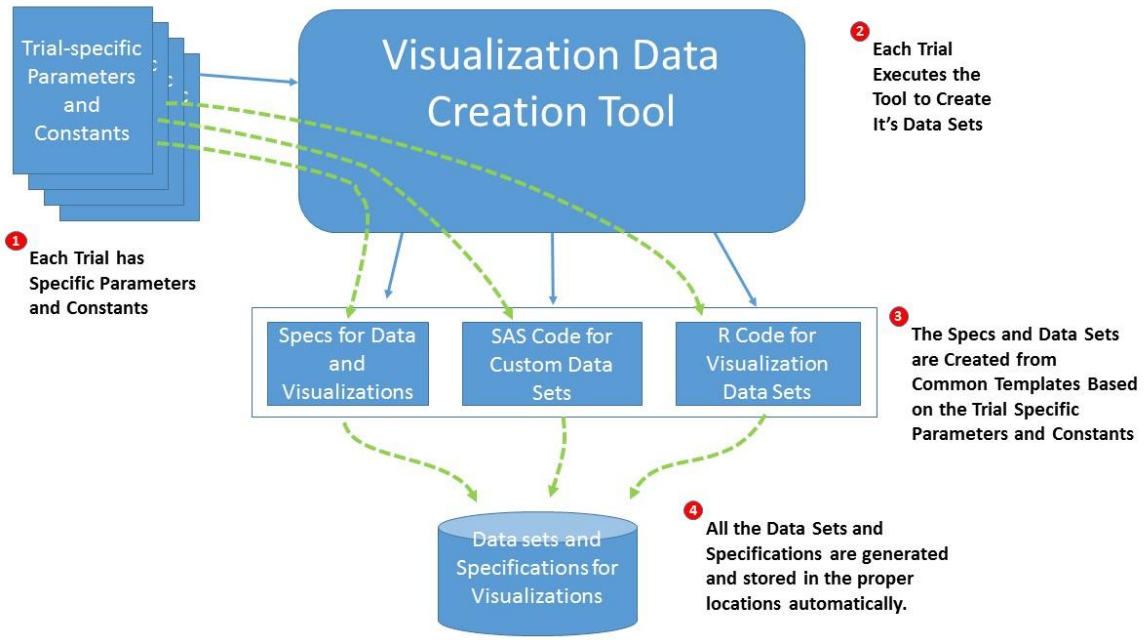
- Utilize a development environment that would organize all of the documentation and programming files and contain a working version that can be modified as the team deems necessary.
- Utilize a development environment that would enable simultaneous multi-user development by providing source code management and version control.
- Utilize a programming environment that would be flexible in running R or SAS code and allow for as much automation as possible, so that both R-centric and SAS-centric programmers could contribute to the project.
- Ensure that the code is flexible and generalizable so that different teams could apply the code to their trial without having to make many modifications.
- Provide one document that programmers could use to enter specific trial information, which would be used by the generalized programs to generate the specifications, data sets, and visualizations required by the Project.

In the next and section, we describe the implementation of the key objectives of the automation tool, providing examples to the reader. We end this paper with a brief summary.

THE SOLUTION: IMPLEMENTATION DETAILS

Figure 1 contains a diagram of the automation tool's inputs and outputs. The tool is a series of R and SAS programs, stored in Github. Programmers of a new study create a fork from the master branch, cloning the series of programs into their preferred working environment. Trial-specific parameters and constants (metadata) are stored in a file that is passed in, and read by the tool's programs. These programs use the metadata to create all specifications and visualizations data sets, and store them in the appropriate locations in the user's working environment.

Figure 1. Diagram of Automation Tool's Inputs and Outputs



To ensure that programs are validated and reproducible for a different study, the software tools for creating the data sets and automating their production are stored in Git.

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency. Many benefits have been realized by using this tool:

• Automates the integration of multi-user development

• Enables multiple projects to be developed simultaneously using the same template while keeping their efforts separate.

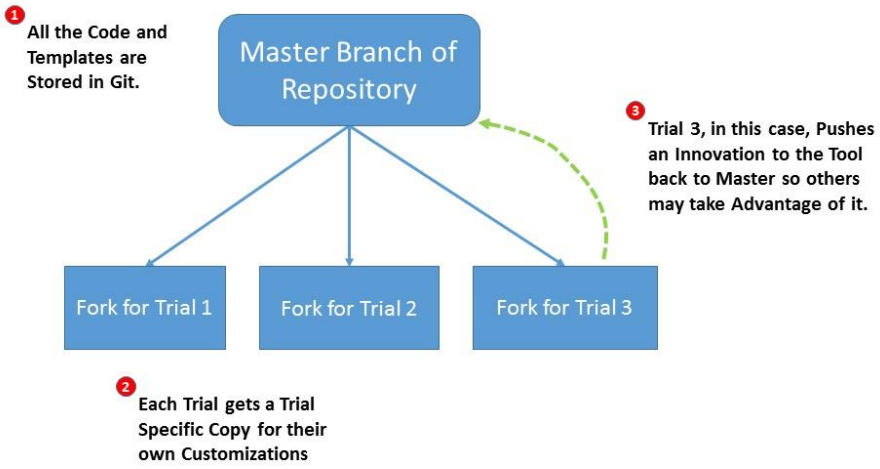
• Consolidates all files for the project into one location.

• Offers file-level and line-level tracking to see what changed, when and why.

• All changes are recorded and undone.

• With the branching mechanism, all changes are independent yet innovations can be promoted into the template and then applied to projects as needed. Figure 2 depicts forks that were created from the master branch for Trials1 – 3. Trial 3 developed additional updates that were later pushed back up to the master branch.

Figure 2. Diagram of Github Master Branch and Trial Forks



No matter how carefully done, when manually typing all the information or editing from the previous codes, mistakes can occur. Also, this process is unnecessarily time consuming. To reduce redundant work and avoid mistakes, a metadata file (study_defs.csv) is used to enter trial specific information. The rest of the programs read this file and produce specifications, data sets, or output based on the content in this file.

The following information is included in the study_defs.csv file:

• The indication/study/analysis related information, such as component codes, study name, program and outputs locations, as well as the visual outputs needed.

• The important information related to the defined visualizations will also be collected. Examples of such information are parameters of interest to plot, time points to plot, label names, treatment specific labels, etc. Anything that is used broadly and consistently across trials but needs to be specific to a trial can be

stored in a variable. Table 1 is an example of the metadata file. The values in the file are fake for the purpose of this example.

Table 1. Example of Study_Defs.csv

PARAMETER	VALUE
COMPOUND	XXX-YY-ZZZZ
STUDY	TRIAL1
CREATE_VISUALIZATION1	TRUE
CREATE_VISUALIZATION2	TRUE
CREATE_VSUALIZATION3	FALSE
TRIAL_COMPOUND	abcd
TRIAL_STUDY	xxxy
TRIAL_ALL_TRT_FULL	Drug A/Drug B
LOC_DIR_ADAM	/aaaa/bb/cccc/eee
LOC_DIR_DATA_OUTPUT	/aaaa/bb/cccc/dd
VAL_VIS1_PAR	PARAMCD1/PARAMCD2/PARAMCD3
VAL_VIS1_PAR_DATASET_NAMES	ADAM1/ADAM1/ADAM3
VAL_VIS1_PAR_PVAL_FILENAMES	filename
VAL_VIS1_VAR_NAMES	SEX AGEGR1 BWGTGR1 ETHNIC BCRPGR1 REGION
VAL_VIS2_PAR	PARAMCD2/PARAMCD4
VAL_VIS2_PAR_DATASET_NAMES	ADAM5/ADAM7

After creating and updating the study_defs.csv file the next step is to compile the list of programs to create and execute. To control the number of programs and prevent overflow of the repository, an R program, make_file_list.R, stores the name of each individual program as well as instructions for whether or not the study team will create the program

In Code reference 1: Visualization 1 Plot File Information, a character vector, called “file”, stores the names of the SAS and R program files needed to create Visualization 1. The study team indicates that Visualization 1 Plot.R will not be created. Other information, such as the name of the folder where these codes will be stored and whether or not the codes will be saved in the production repository is described as well:

CODE REFERENCE 1: VISUALIZATION 1 PLOT FILE INFORMATION

```
area <- "VIS1"

file <- c("adam_for_vis1.sas",
          "Vis1_Dataset_Specification.Rmd",
          "Vis1_Visualization_Specification.Rmd",
          "create_custom_vis1_dataset.R",
          "Visualization 1 Plot.R")

directory <- "visualization1"
build <- c(TRUE, TRUE, TRUE, TRUE,
                FALSE)                          #Do not create last file
xfer_to_prd <- TRUE
process_with_brew = TRUE
```

```
file_list <- rbind(file_list, data.frame(area, directory, file, build,
          xfer_to_prd))
```

Similarly, Code Reference 2: Visualization 2 Plot File Information lists the codes needed for Visualization 2. The rbind function appends the new list of programs to existing list of files from Visualization 1:

CODE REFERENCE 2: VISUALIZATION 2 PLOT FILE INFORMATION

```
area <- "VIS2"

file <- c("vis2_data.sas",
          "ir_vis2_data.sas",
          "vis2_data_spec.Rmd",
          "vis2_visualization_spec.Rmd")

directory <- c("visualization2",
          file.path("visualization2", "ir"),
          "visualization2",
          "visualization2")

build <- TRUE

xfer_to_prd <- TRUE
process_with_brew = TRUE
file_list <- rbind(file_list, data.frame(area, directory, file, build,
          xfer_to_prd))
```

In Code Reference 3: Visualization 3 Plot File Information, it is indicated that the entire list of programs related to Visualization 3 will not be created, as the study team did not request this visualization.

CODE REFERENCE 3: VISUALIZATION 3 PLOT FILE INFORMATION

```
area <- "VIS3"

file <- c("vis3_data.sas",
          "vis3_data.sas",
          "vis3_ComponentHistogramAnimatedVis.Rmd",
          "vis3_ComponentHistogramStaticVis.Rmd",
          "vis3_animation.R",
          "vis3_PlotAnimationFrame.png",
          "vis3_PlotStaticFrame.png")

directory <- "visualization3"

build <- FALSE
xfer_to_prd <- FALSE
process_with_brew = TRUE
file_list <- rbind(file_list, data.frame(area, directory, file, build,
          xfer_to_prd))
```

Table 2 shows the data frame that describes the final list of programs that will be executed by the study team. These are all the programs where the user specified build = TRUE, as well as “helper” programs that read and process the study_defs.csv to create global variables that will be used by the programs to create the data sets and visualizations.

Table 2. Example of file_list

AREA	DIRECTORY	FILE	BUILD	XFER_TO_PRD
UTIL	.	create_data.R	FALSE	FALSE
UTIL	includes	helpers.R	TRUE	TRUE
UTIL	includes	read_study_defs_csv.R	TRUE	TRUE
UTIL	includes	study_defs.csv	TRUE	TRUE
UTIL	includes	process_study_defs_csv.R	TRUE	TRUE
VIS1	visualization1	adam_for_vis1.sas	TRUE	TRUE
VIS1	visualization1	Vis1_Dataset_Specification.Rmd	TRUE	TRUE
VIS1	visualization1	Vis1_ Visualization_Specification.Rmd	TRUE	TRUE
VIS1	visualization1	create_custom_vis1_dataset.R	TRUE	TRUE
VIS1	visualization1	Visualization 1 Plot.R	FALSE	TRUE
VIS2	visualization2	vis2_data.sas	TRUE	TRUE
VIS2	visualization2/ir	ir_vis2_data.sas	TRUE	TRUE
VIS2	visualization2	Vis2_data_Spec.Rmd	TRUE	TRUE
VIS2	visualization2	Vis2_ Visualization_Specification.Rmd	TRUE	TRUE
VS3	visualization3	vis3_data.sas	FALSE	FALSE
VS3	visualization3	vis3_plot_dataset_spec.Rmd	FALSE	FALSE
VS3	visualization3	vis3_ComponentHistogramAnimatedVis.Rmd	FALSE	FALSE
VS3	visualization3	vis3_ComponentHistogramStaticVis.Rmd	FALSE	FALSE
VS3	visualization3	vis3_animation.R	FALSE	FALSE
VS3	visualization3	vis3_PlotAnimationFrame.png	FALSE	FALSE
VS3	visualization3	vis3_PlotStaticFrame.png	FALSE	FALSE

For each R program to be run, the study_defs.csv file will be read and processed. The read_study_defs_csv.R and process_study_defs_csv.R programs create variable assignments that can be used throughout the program. For example, if study_defs.csv file is the same as that in Table 1. Example of Study_Defs.csv, the following assignments will be made: CREATE_VISUALIZATION 1 <- TRUE, LOC_DIR_ADAM_DATA <- “/aaaa/bb/cccc/dd”“, VAL_VIS1_PAR <- ”PARAMCD1/PARAMCD2/PARAMCD3“, etc. In Code Reference 4: Example of study_defs.csv implemented in R Code: Example of study_defs.csv implemented in R Code, the strsplit function can be used to break the VAL_VIS1_PAR string into a vector of strings, where the ith element of the vector is the ith element of VAL_VIS1_PAR, delimited by”/“.

CODE REFERENCE 4: EXAMPLE OF STUDY_DEFS.CSV IMPLEMENTED IN R CODE

```
source(file.path("includes", "read_study_defs_csv.R"))
source(file.path("includes", "process_study_defs_csv.R"))

...
adsl <- file.path(LOC_DIR_DATA_ADAM, "adsl.sas7bdat")
ep_files <- lapply(file.path(LOC_DIR_DATA_ADAM, file_dat$filename),
read.sas7bdat)
```

```
  paramlist <- strsplit(VAL_VIS1_PAR, "/")[[1]]
```

Similarly, when creating specifications for Visuals or data sets, R markdown can be used to write generalized instructions. Markdown is a simple markup language used to create documentation. In Code Reference 5a: `Rmd Code` and Code Reference 5B: `Rmd HTML output`, the information from study_defs.csv is read in to an R markdown program to create specific instructions. Utilizing this platform provides programmers with the flexibility of creating customized documentation files and yet allow the Rmd program to automatically update the project specific constants in the documentation. Once any updates are made, the program automatically converts the markdown files into HTML pages that serve as the documentation for the data sets and visuals. Because the markdown files are text based, character processing can be done to automate updates. In addition, as they are stored in Git, line level tracking of changes is readily available.

CODE REFERENCE 5A: RMD CODE

```
.dat_sets <- sapply(strsplit(VAL_VIS1_PAR_DATASET_NAMES, "/")[[1]], tolower)
.dat_params <- strsplit(VAL_VIS1_PAR, "/")[[1]]

cat(paste("- Only use records from ", .dat_sets , " where `PARAMCD is " ,
.dat_params , "`, `ANL01FL = 'Y'`, `AVISITN = 8`, `APERIOD = 1`, and `DTYPE =
''`.", collapse = "\n"))
```

CODE REFERENCE 5B: RMD HTML OUTPUT

```
• Only use records from adam1 where PARAMCD is PARAMCD1, ANL01FL = 'Y',
AVISITN = 8, APERIOD = 1, and DTYPE = ''. • Only use records from adam1 where
PARAMCD is PARAMCD2, ANL01FL = 'Y', AVISITN = 8, APERIOD = 1, and DTYPE = ''.
• Only use records from adam3 where PARAMCD is PARAMCD3, ANL01FL = 'Y',
AVISITN = 8, APERIOD = 1, and DTYPE = ''.
```

Every generalized SAS program will refer to a macro variable called csvpath. This macro variable stores the location of the study_defs.csv file. The tool assigns csvpath before it executes the SAS code. In Code Reference 6: `Example of study_defs.csv implemented in SAS Code`, the SAS program uses csvpath to read in study_defs.csv and create global macro variable assignments where PARAMETER is the name of the macro variable and VALUE is the macro variable's value. Then the SAS program uses these macro variable assignments to generate specific outputs.

CODE REFERENCE 6: EXAMPLE OF STUDY_DEFS.CSV IMPLEMENTED IN SAS CODE

```
proc import datafile="&csvpath" out=defs dbms=csv replace;
      guessingrows=1000;
run;

data _null_;
      set defs(where=(value ne ""));
      call symputx(parameter,value);
run;

libname adam "&adamloc";
libname outlib "&outpath";
%macro makedata();
%let aa=1;
%do %while(%scan(&paramlist,&aa,/) ne );
      %let bb=%scan(&VAL_VIS2_PAR,&aa,/);
      %let cc=%scan(&VAL_VIS2_PAR_DATASET_NAMES,&aa,/);

data &bb;
```

```
      set adam.&cc(where=(paramcd="&bb" and &strt<=avisitn<=8 and
      avisitn=int(avisitn) and anl02fl="Y" and aperiodc="Treatment Period"
      and dtype IN ("", "NRI")));
run;
%end;
%mend;
%makedata();
```

A program called create_data.R sources the make_file_list.R program, and then sources the SAS, R, and Rmd programs that are defined in the file_list data frame (Table 2. Example of file_list).

- R programs are sourced using the source() function.
- Each SAS program has a line of code appended to the beginning. The code included at the beginning of the program is %let csvpath=%str("/.../.../.../study_defs.csv") where the content in quotations is the location of the study_defs.csv file.
- The SAS program imports study_defs.csv and creates global macro variables to be used throughout the program.
- Each Rmd file is rendered using the rmarkdown package into an HTML file.
- The brew function from the "brew" library populates the headers of the programs with information from study_defs.csv. Brew reads the R/SAS program as a text file, evaluating variables between the strings "." (Code Reference 7A: Excerpt of R program header before it is evaluated by BREWand Code Reference 7B: Excerpt of R program header after it is evaluated by brew).

CODE REFERENCE 7A: EXCERPT OF R PROGRAM HEADER BEFORE IT IS EVALUATED BY BREW

```
# PROJECT NAME (required)      :  Efficacy Data Set Generation for compound: <%=COMPOUND%>
#                                 STUDY<%=STUDY%>
# SOFTWARE/VERSION(required):  <%R.version$version.string%>
# DATA INPUT                   :  <%=LOC_DIR_ADAM>
# OUTPUT                       :  <%=LOC_DIR_DATA_OUTPUT>
# PROGRAM AUTHOR               :  JANE DOE
```

CODE REFERENCE 7B: EXCERPT OF R PROGRAM HEADER AFTER IT IS EVALUATED BY BREW

```
# PROJECT NAME (required)      :  Efficacy Data Set Generation for compound: abcd
#                                 STUDY TRIAL1
# SOFTWARE/VERSION(required):  R version 3.5.2 (2018-12-20)
# DATA INPUT                   :  /aaaa/bb/cccc/eee
# OUTPUT                       :  /aaaa/bb/cccc/dd
# PROGRAM AUTHOR               :  JANE DOE
```

CONCLUSION

The need to leverage multiple programming platforms (SAS and R) in clinical drug development programs, for producing standard graphical and text outputs across multiple therapeutic areas, inspires creation of tools for automation that could be broadly used by study teams. The task to generate

interactive efficacy visualizations for multiple trials posed difficulties to the study teams due to the perceived differences across the projects. However, applying a single tool that capitalizes on the similarities creates a generalized solution that makes automation possible and reduces programming efforts. Along the way, new technologies such as Git, an automation environment that can handle R and SAS programs seamlessly, and generalization of the production of data sets so that they can be applied to any trial were introduced to increase the study team organization and efficiency.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

<Anastasia Alexeeva>
<Eli Lilly and Co)>
<alexeev_anastasia@lilly.com>

<William Martersteck>
<Eli Lilly and Co)>
<martersteck_william@lilly.com>

<Mei Zhao>
<Eli Lilly and Co>
<zhao_mei@lilly.com>

BASIC INSTRUCTIONS

WRITING GUIDELINES

Writing style

- Use active voice. (Use passive voice only if the recipient of the action needs to be emphasized.) For example:

  The product creates reports. (active)
  Reports are created by the product. (passive)

- Use second person and present tense as much as possible. For example:

  You get accurate results from this product. (second person, present tense)
  The user will get accurate results from this product. (future tense)

- Run spellcheck, and fix errors in grammar and punctuation.

Citing references

All published work that is cited in your paper must be listed in the REFERENCES section.

If you include text or visuals that were written or developed by someone other than yourself, you must use the following guidelines to cite the sources:

- If you use material that is copyrighted, you must mention that you have permission from the copyright holder or the publisher, who might also require you to include a copyright notice. For example: "Reprinted with permission of SAS Institute Inc. from *SAS® Risk Dimensions®: Examples and Exercises*. Copyright 2004. SAS Institute Inc."
- If you use information from a previously printed source from which you haven't requested copyright permission, you must cite the source in parentheses after the paraphrased text. For example: "The minimum variance defines the distance between cluster (Ward 1984, p. 23)

TIPS FOR USING WORD

These instructions are written for MS Word 2007 and MS Word 2010. The steps are similar for MS Word 2003.

To select a paragraph style

1. Click the HOME tab. The most common styles in your document are displayed in the top right area of the Microsoft ribbon. If you don't see a style that you want, click the slanted down arrow at the bottom right corner of the Styles area, and scroll through the list. The main styles for this template are headings 1 through 4, PaperBody, and Caption. Avoid using other styles.
2. To change a paragraph style, click the paragraph to which you want to apply a style, and then click the style that you want in the ribbon.
3. PaperBody (used for most text) is automatically applied when you press Enter at the end of any heading style or the Caption style.

To insert a caption

1. Click **REFERENCES** on the main Word menu.
2. Click **Insert Caption**.
3. Select the **Label** type that you want.
4. Click **OK**.

To insert a cross-reference

1. Click **REFERENCES** on the main Word menu.
2. Click **Cross-reference**.
3. In the **Reference type** list box, select Heading, Figure, Table, Display, or Output.
4. For a heading:
   a. In the **For which heading** list, select the heading that you want.

b. From the **Insert reference to** list, select **Heading text.**

5. For a figure, table, display, or output:

   a. In the **For which caption** list, select the caption that you want.

   b. From the **Insert reference to** list, select **Only label and number**.

To insert a graphic from a file

1. Click **INSERT** on the main Word menu.
2. Click **Picture**.
3. In the Insert Picture dialog box, navigate to the file that you want to insert.
4. When the name of the file that you want to insert is displayed in the **File name** box, click **Insert**.