

Computers & Language / Language Generation

The iGraph-LITE Book

LEO FERRES *University of Concepción*



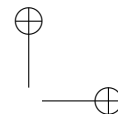
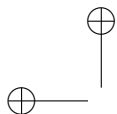
UNIVERSIDAD DE CONCEPCIÓN

Department of Computer Science

Faculty of Engineering

Edmundo Larenas 215

Concepción · Chile

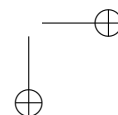
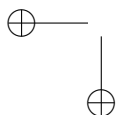


Internet page <http://www.inf.udec.cl/~leo/nlcm.html> contains current information about this book and its associated algorithms.

Copyright © 2009 by Leo Ferres

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher, except that the official electronic file may be used to print single copies for personal (not commercial) use.

Zeroth printing (revision 0), 21 September 2009.



PREFACE

A picture is worth ten thousand words.
— ANONYMOUS

This is a handbook about iGraph-LITE, an extensible tool that automates the process of generating natural language descriptions of statistical graphs produced by plotting software such as Excel, OpenCalc, SPSS and others. By using iGraph-LITE, you will be able to automatically generate descriptions of graphs in *batch* mode, without having to do the time-consuming cycle of looking at a graph, extracting the relevant information, crafting a paragraph and finally putting it into a formatted form for publishing in different media (e.g. HTML, PDF or even just plain text). In fact, given new laws such as Section 508 of the United States Government and the Charter of Rights and Freedoms in Canada, iGraph-LITE should be able to greatly help companies, organizations and even individuals during the authoring of accessible information, particularly those organizations that produce an large number of graphs per day.

This handbook is intended for both: people who will make use of the utility itself (users), and people who may want to extend it in some way (hackers). Everything there is to be known about iGraph-LITE is here, including detailed documentation on the source code, which should hopefully clarify the sometimes awkward behavior of the program, and enough information to allow hackers to quickly solve the problems this behavior causes. Thus, if the reader just wants to use iGraph-LITE for a set of simple graphs, then reading the first ten or so pages of this manual will be sufficient; on the other hand, if the reader/user wants to account for more complex graphs (and this is highly likely), he or she will surely have to venture into the innards of the source code of iGraph-LITE and change, extend or improve the current state of the program. This is fine, indeed encouraged, to make iGraph-LITE a better tool.

iGraph-LITE’s name has a short history that I will tell briefly. The name was thought of before the advent of the ubiquitous “i” now so common in technological gadgets (*iPod*, *iPhone*, and even another *iGraph!*)¹. The first version of iGraph-LITE was called *inspectGraph*, and it was only a proof-of-concept system written back in 2004 at the Human-Oriented Technology Laboratory, at Carleton University in Ottawa, Canada. At the time, I was just researching what exactly such a system should do, gathering requirements, designing its

¹<http://igraph.sourceforge.net/>

different logical parts and, although I have always been sure of the potential of such a tool, trying to find some *telos* to it, some practical function that I could proud of. This last part happened in 2005, when my colleague Avi Parush and I won a Health Technology Exchange grant to develop iGraph-LITE as an Assistive Technology to help blind and visually-impaired persons gain access to visual information encoded in digital format. For this project, and probably given the natural trend of languages towards conciseness, the name *inspectGraph* was shortened to iGraph plain, that is without the LITE part. This was so because the interface to the information was intended to be a full-fledged natural language question-answering system, using Jabber and XMPP over the Google Talk client to “ask” questions about a graph in dialog form. It soon became evident that this interface was perhaps not the most felicitous one, particularly given the kinds of tools that blind and visually-impaired people were used to working with. Since I was after a sizable impact for our target community, I decided that the iGraph-LITE team was going to simplify the interface and make interaction more like that found in state-of-the-art screen-reading technologies such as JAWS or DOLPHIN. We thus concentrated more and more on the content of the language to be used by the description engine, and on designing the command-driven interface for retrieving information that had been left out by the description engine mentioned above. Since this was a simpler version of the early pomposity of iGraph (for those who do not work with dialog-driven natural language interfaces, they are a *really* hard to tame), the name iGraph-LITE, using the “Lite” neologism (instead of the more proper “Light”, maybe as a bow and hidden homage to Description Logics (as in DL-Lite)) was proposed and accepted. This is the name that remains today.

Perhaps a word should also be said here about the choice of the implementation language, Microsoft’s C#. I know some of the more purist computer scientists and hackers out there will cringe at my choice. They will be right at several levels, alas. But I have reached a stage in my life in which the boring Language Wars do not to affect me very much. Indeed, I could have programmed iGraph-LITE in several other, more “highly esteemed” languages such as C++ (!), Java, Python, or Lisp; and truth be told, different versions of the system *were* implemented in two of those languages before finally settling down on C#. C++ was too low-level and Lisp did not have some critical Windows Excel Application Programming Interface (API), and they were thus never a choice for this project. *inspectGraph* was implemented first in Java, and the first complete version of iGraph-LITE implemented in both Java and Python. Besides the obvious “overkill” issue of using C++, reasons why I rejected the Java and Python implementations are of different kinds, and I don’t have time to go into them here. Suffice it to say that in programming, not considering the usual and sad dogma of programmers and their pet languages, you usually want the best tool for the job, and C# was it. I trust that users will not care how iGraph-LITE works under the hood, and that real hackers will not care what syntax they use when working with, fixing or improving the system. If they need an open source compiler, hackers can always use Mono² or the free (closed source) version of Microsoft’s C# com-

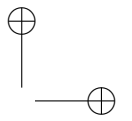
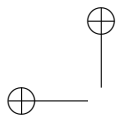
²www.mono-project.com/

piler (<http://www.microsoft.com/express/vcsharp/>)... as you see, there are several options, so please do not let philosophy get in the way of contributing. I, for one, haven't—and this book is being written using L^AT_EX, with Emacs, on a machine running Ubuntu. . .

The book is structured as follows: it has two parts—the user manual part and the hacker part. In the first part, I will mostly describe how to make iGraph-LITE function properly, and how to read its output, fix simple potential errors in graphs, etc. The second part is a detailed exploration of the code itself: every class, and every method are explained. This will allow some nice hacker to understand the design with relative speed and get coding much sooner, I hope, than if this documentation did not exist. Besides, I think some parts of iGraph-LITE deserve the explanation, if only to humor this author with respect to the indescribable feeling of awe that some of these algorithms inspired in him. For this self-indulgence, I apologize.

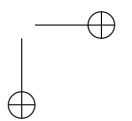
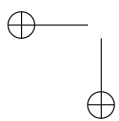
I wish to thank the many people who, directly or indirectly, have been involved in the making of iGraph-LITE and that have helped me formulate the details of the system as it currently stands, which has very little in common with the early *inspect*-Graph of 2004. Among them, I would like to mention Petro Verkhogliad, Livia Sumegi, Gitte Lindgaard, Avi Parush at Carleton University; Antoine Chrétien, Martin Lachance, Louis Boucher, Stacy Bleeks, Krista Kowalchuk, Pamela Best, and Ivan Gervais, at Statistics Canada. My Work at Carleton University and the University of Concepción has been generously supported by the Human-Oriented Technology Laboratory, Statistics Canada, Cognos/IBM, the National Science and Engineering Research Council of Canada, the Health Technology Exchange of Ontario, and by the Department of Computer Science at the University of Concepción.

— L. F.
 University of Concepción
 Chile, March 2009



CONTENTS

1	The Handbook	2
1.1	Introduction	2
1.2	Installation instructions	2
1.3	Running iGraph-LITE	3
1.4	Reading the iGraph-LITE output	6
1.5	Formatting graphs correctly in the graphing application	8



CHAPTER 1

THE HANDBOOK

1.1 INTRODUCTION

iGraph-LITE is a software tool that provides textual descriptions to graphs composed using Microsoft Excel, particularly those produced for Statistics Canada’s main online publication: “The Daily”¹. iGraph-LITE also produces several kinds of intermediate outputs that may be of interest/importance for several parties including statistical agencies, developers, accessibility professionals, and business analysts, to name but a few.

The iGraph-LITE program can be seen in several different lights. We will first explain how to install and run the current version of iGraph-LITE, and then we will explain how to read the output, modify its descriptions, check for the accuracy of the description and so on.

it is important to emphasize that at this stage, iGraph-LITE is a highly experimental application, and as such, it will (possibly) have errors that could not be caught in the laboratory, even after testing it thoroughly on several hundreds of the graphs produced by “The Daily”. This notwithstanding, we believe that the system, together with the conscientious creation of graphs, will provide a good first attempt for the highly sought after reality of accessibility to graphs.

1.2 INSTALLATION INSTRUCTIONS

In a nutshell, iGraph-LITE is a system that takes MS Excel® files containing a graph as input and outputs a natural language description of this graph, among other outputs that we will discuss below. Although the first few versions of iGraph-LITE were two different applications (the graph creator and the generator), these systems have now been merged into one system, hopefully facilitating its use.

iGraph-LITE can be downloaded from <http://www.inf.udec.cl/~leo/igraph.html>, the latest version will always be available from <http://www.inf.udec.cl/~leo/igraph-latest.zip>, but it is important to check for new versions of it regularly, since this is still a work in progress.

The installation is very simple. Once the file has been downloaded, it only has to be unzipped in some directory, and it is ready to be used.

¹<http://www.statcan.gc.ca/dai-quo/index-eng.html>

1.3. RUNNING IGRAPH-LITE

3

1.3 RUNNING IGRAPH-LITE

The test downloadable version of the command line tool is packaged as a Microsoft Windows executable, `cli.exe`. Running `igl.exe` with the `-h` option produces the following output:

```
This is iGraph-Lite, Version 1.2.3589.28966 (10/29/2009 4:05:33 PM)
Log level set to: WARN
```

```
Usage: igl.exe [OPTIONS] Directory
```

Options:

```
-g, --gif           Export graph to gif file
-?, -h, --help     Show this message and exit
-c, --csv-file=VALUE Load a csv file
-f, --input-file=VALUE Specify the directory
-l, --log-level=VALUE How much iGraph will say (0 to 6)
-x, --xml          Export graph to XML file
-o, --owl          Export graph to OWL 2.0 file
```

Options that take values may use an equal sign, a colon or a space to separate the option from its value.

If no directory is specified, the current directory is used.

Let us analyze this for a bit. Assuming the installation has succeeded, and the only reason why it should not is if Excel is not installed, or the .Net framework is not the correct one (errors which iGraph-LITE will detect and report), iGraph-LITE writes a welcome message with the version number, `Version 1.2.3589.28966 (10/29/2009 4:05:33 PM)`, in this case. This is an important message, since it will tell the user the version being run, which will help with the debugging process.

The date and time is the day and time that iGraph-LITE was built in our lab, which also provides versioning information.

The usage of iGraph-LITE is also quite simple. We simply can run it with the following command:

```
> igl.exe
```

in which case it will read all the Excel files in the current directory. Basically, it will read all the Excel files that are present in the current directory, or the equivalent `igl.exe .` (run iGraph-LITE in the current directory). There are two other ways to process graphs. First, we may explicitly say the directory of the `xls` or `xlsx` files, as in:

```
> igl.exe c:\ig\graphs
```

in which case the system will attempt to process the graphs present in `c:\ig\graphs`. We may also use relative paths in iGraph-LITE, but in the Windows platform this is not always recommended. In any case, supposing we are in the `c:\ig` folder, we may issue the following command, which will be equivalent to `igl.exe c:\ig\graphs`:

```
> igl.exe .\graphs
```

where the dot “.” means “current directory”. If we were positioned at `.\graphs\lines`, we could also issue the `iGraph-LITE` command as

```
> igl.exe ..\
```

where the two periods “..” means “go back one directory”, and so on.

By request from Statistics Canada, and their workflow, we have also allowed to input a comma-separated file (using the `csv` option). If this option is used, `iGraph-LITE` will build a list of files to be processed from this file. The format of said file is:

- path to the file (absolute),
- the lanaguage of the file (either English or French, thoughh this can be extended) and finally
- the title.

so, a comma-separated file with just one entry may look like this:

```
c:\graphs\lines\c080808a.xls,English, "Some title"
```

Notice that if the `csv` option is used, then there will be no attempt to dynamically find the title of the graph in the Excel file. The title will be simply set to what is specified in the comma-separated file.

Sometimes it is handy to generate or output information about one Excel file, instead of a whole directory of them. This is helpful for debugging, for instance. In order to do this, `iGraph-LITE` can be run with the `f` option, in the following way:

```
> igl.exe -f c080808a.xls
```

which basically tells `iGraph-LITE` to only process the `c080808a.xls` file.

These are the options we can use to input files into `iGraph-LITE` for further processing. We may call these the “input” options. There are two other kinds of options that `iGraph-LITE` may receive. We may call these the “utility” and the “writing” options. Utility options are used to either know more about `iGraph-LITE`, or to debug its processing (the `--help` option discussed above is an example of these utility commands).

Another utility command is the `-l` option, short for `--log-level` option. The `--log-level` option sets how much `iGraph-LITE` will say about what it is doing during processing. This is very important if `iGraph-LITE` is processing many files and we cannot know why some of them are failing. The `l` option takes an integer between 0 and 6 as parameter, 0 being *do not say anything at all*, and 6 meaning *say as as much as you can*. The log level is set as follows:

```
> igl.exe -l=0 -f c080808a.xls
```

or

1.3. RUNNING IGRAPH-LITE

5

```
> igl.exe -l 6 -f c080808a.xls
```

Notice the lack of the equals sign (“=”) in the second example: options with values other than `true` and `false` can be entered with or without the equals sign. We will discuss logging in detail in Section 1.4 below.

The final type of options are the “writing” options. These options generate other kinds of input other than the default HTML file with the description and the table with the data. They are the `--gif`, `--xml` and `--owl` options. Of these, the latter is not yet implemented, but should be relatively soon.

The `--gif` or `-g` option simply outputs a GIF representation of the graph. The file will be named as `c080808c_1_1.gif`, or, more generally,

```
<excelFileName>_<excelWorksheetIndex>_<worksheetGraphID>.gif
```

where `c080808` is the name of the Excel file where the graph was found. The first `_1` is the worksheet within that file in which the graph was found, and the last `_1` is the index of the graph in the collection of graphs appearing in that worksheet. Thus, for example, a graph with the name `c080808_2_4.gif` is the GIF (raster image) file of the fourth graph in the second worksheet of Excel file `c080808.xls`.

The next writing option is `--xml` (or `-x`) writes an XML version of the graph. The generated XML document is not a one-to-one correspondence to the graph in the Excel file, but is constructed from the post-processing that iGraph-LITE does on the information obtained from Excel, including cleaning the titles, series, categories and the like. To generate the XML representation, `igl.exe` is called with

```
> igl.exe -x .\graphs
```

The name of the file follows the same convention as with the GIF file, except that the file extension will be, not surprisingly, `.xml`.

Although we may combine several writing options, or all of them, not surprisingly, we cannot do the same with input options. We may only specify one of the input options. If we inadvertently put in two, only the last input option will be processed. For instance,

```
> igl.exe -x -g -f c080808a.xls .\graphs
```

The `--owl` option, when implemented, will behave exactly as both the `-g` and `-x` options behave, except that the generated file will have the extension `.owl`. OWL is the Web Ontology Language, the computational implementation of the SROIQ(D) Description Logics, which underpins the Semantic Web. then iGraph-LITE will generate GIF and XML files of all the graphs in the `.\graphs` folder, but will not execute the `-f` option.

This concludes how to run iGraph-LITE. We now turn to how to read iGraph-LITE messages.

1.4 READING THE IGRAPH-LITE OUTPUT

iGraph-LITE is able to produce a lot of output that helps both understanding its processing, and also making sure that everything is going according to plan. Even if it is one single graph we are processing, the information that iGraph-LITE outputs will help ensuring the correct processing of that graph.

iGraph-LITE has 7 settings (0 to 6) for “verbosity”, that is, how much iGraph-LITE will say during processing. We set this option, as discussed above, with the `-l=<integer>` option, where the integer is greater or equals to 0, and lower or equals to 6. The higher the number, the more information iGraph-LITE will output to the console. We now describe each of these levels.

We can see the log level information below the iGraph-LITE banner. In the case above, the system says `Log level is set to: <log level>`, in this case `WARN`. We will discuss some of the log levels, namely 0, 3 and 6, which are the most common ones. If the 1 value is left unspecified, log level 3 is the default value with which iGraph-LITE starts.

- Logging level 0 (log level set to: OFF): There will be no information output by the logger. However, there may still be messages that are fatal to the application, and those will be output, in red, to the console. Examples of these fatal errors are:

1. Log level set to: OFF
 Error: Directory does not exist.
 You said: test
 Remember to add quotes if directory contains spaces.
2. Log level set to: OFF
 Error: The specified file was not found.
 You said: nonexistentfile
3. [DEBUG] Attempting to parse CSV file.
 [DEBUG] Parsed 0 entries, 93 were ill-formed.
 Error: CSV file found, but no Excel files in it.
 You said: focal\c080220e.xls

Notice that number 3 is special. In this case, anything can be a comma-separated file. However, unless the files have the proper formatting discussed above, they will not be processed. The information in the `DEBUG` tells us that for this potential CSV file, there were a total of 93 entries, and 0 were well-formed. The error defaults to being unable to find the graph files. Also in the last example the log level was set to 6, or `ALL` and that is why we obtain much more information.

- Logging level 3 (Log level set to: WARN): This logging level only shows warning and error information (but not debug information, see below), problems that are not critical, but should be removed if possible (in the case of warnings) or problems that are indeed critical for a graph (such as an empty series), but that does not make iGraph-LITE stop necessarily. iGraph-LITE processing will not stop on account of these errors, but it is highly suggested that the problems or issues (usually an ill-formed

1.4. READING THE IGRAPH-LITE OUTPUT

7

graphs) are solved before proceeding. Below is just one example of this log level information.

```

1. [ WARN] Couldn't find the title of the graph.
    Setting it to: "none".
   [ WARN] Couldn't find the title of the value axis.
    Setting it to: "none".
   [ WARN] Category axis name not found.
    Setting it to: "none".
   [ WARN] Orphan textbox: Canada = 27,5
    
```

Notice that all the information coming from the log will have the severity of the problem in-between brackets, and then an explicit cause of the warning or error. In the case above, the warnings include an “orphan textbox” (a textbox that does not seem to fulfil a recognized semantic function). These are prepended with [WARN].

- Log level 6 (Log level set to: ALL): This logging level will show all kinds of information that is relevant mostly for debugging some iGraph-LITE problem. There is a considerable amount of information being output to the console and this will help greatly with the search for problems... or indeed for mere curiosity. Below is a transcript of a whole session, which we discuss immediately after it.

```

C:\igl>igl.exe -l 6 -f focal\c080220e.xls
This is iGraph-Lite, Version 1.2.3593.22457 (11/2/2009 12:28:34 PM)
Log level set to: ALL
[DEBUG] Communication with Excel started successfully.
This is Alpha-Lexis, Version 1.0.
Processing file: C:\igl\focal\c080220e.xls
[DEBUG] Graph ID: c080220e_1_1
[DEBUG] Plot area found, properties are posX=8, posY=38, h=183, w=170.
[DEBUG] Textbox found: id=0, (posX=0, posY=6, h=28.5, w=234.75).
[DEBUG] Textbox found: id=1, (posX=74.25, posY=246, h=13.5, w=144.75).
[DEBUG] Textbox found: id=2, (posX=0, posY=268.5, h=12.75, w=234.75).
[DEBUG] Textbox found: id=3, (posX=171, posY=169.5, h=21.75, w=43.5).
[DEBUG] 4 legal text box(es) found.
[DEBUG] Legal series found: ID=0, name=Rate, type=2, 9 points.
[DEBUG] 1 legal series found and added to the list.
[ WARN] Couldn't find the title of the graph.
    Setting it to: "none".
[ WARN] Couldn't find the title of the value axis.
    Setting it to: "none".
[DEBUG] Legal value axis found: title is none, min=0, max=50, step=5.
[ WARN] Category axis name not found.
    Setting it to: "none".
[DEBUG] Attempting cleaning of graph: C080220E_1_1
[DEBUG] Applying textbox cleaning algorithms.
[DEBUG] Probable main title recognized: Firearm-related...
[ WARN] Orphan textbox: Canada = 27,5
[DEBUG] Probable value axis title recognized: Rate of victims...
[ INFO] Probable footnote recognized: 1. Data for Vancouver...
[DEBUG] Applying series cleaning algorithms.
[DEBUG] All series seem OK. Bravo!
[DEBUG] Applying category axis cleaning algorithms.
[DEBUG] [Hamilton] [Quebec] [Ottawa] [Edmonton] [Montreal]...
[DEBUG] Probable primary category: UNDEF
    
```

```
[DEBUG] No secondary axis.
[ INFO] Successfully released the Excel handle. This is awesome.
Done.
```

Let us analyze a whole iGraph-LITE log using the highest log level in detail.

After successfully starting the system, the first debug statement informs that the Excel libraries are present in the system and ready to be used. After this, no matter what the debug level is, the system will output the path of the file it is currently processing, prefixed by “Processing file:” and the directory and file names. Every graph in the file is given a unique identifier built by the index of the graph in the worksheet graph collection (in case there is more than one graph in it), the number of the worksheet and the filename, this is the “Graph ID” information. In this case, the graph is given the “c080220e.1_1” identifier.

iGraph-LITE will start the real processing of the graph after the unique identifier has been assigned. The first processing instructions have to do with recognizing the different parts of the input graph and building an internal representation of it. Information such as the size of the plot area, the number of textboxes and their contents, the number of series and their values is gathered. This is information that will prove useful when generating the descriptions. We must keep in mind that not all information available about the graph is actually desirable. In this case, the color of the titles, or whether they are italics or not is not really relevant for further processing. This notwithstanding, it is not extremely complex to add this information if we find the need for it in the future.

The next few lines attempt at finding the titles of the graph in question. In this case we get a warning (unlike the DEBUG statements before), because the graphs do not have a formal title, and these titles will therefore have to be inferred from the context, a process that will happen later, when the cleaning algorithms come into effect, a few lines ahead.

The next few lines (some of which have been clipped for ocular pleasure) are the output of the cleaning algorithms mentioned in the previous paragraph. We will come back to them in the next section, suffice it to say that some of them will clean the series values (maybe there are nulls or just zeros), the titles, the categories (years, months, etc.) and so on.

If all goes well, the Excel handle is released and the system is done.

1.5 FORMATTING GRAPHS CORRECTLY IN THE GRAPHING APPLICATION

iGraph-LITE cannot work properly if the graphs do not follow a few simple conventions. Consistency and preservation of the original data in any object is of much importance, and iGraph-LITE is not the exception. Below are quite a few examples of critical problems for iGraph-LITE, and indeed any other application that would work with such representations.

Example 1: Keeping the languages constant. Graph c080201b is a graph in English. However, the series within the graph have been given French names.

1.5. FORMATTING GRAPHS CORRECTLY IN THE GRAPHING APPLICATION

iGraph-LITE cannot tell the difference and happily assigns French names to the series.

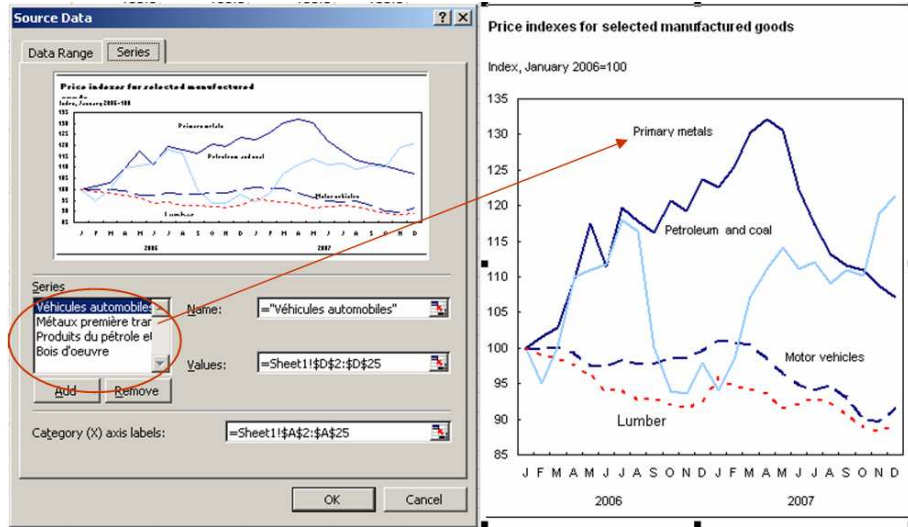


Figure 1.1: Keeping the language constant

The generated text for graphs with this problem goes along the following lines:

The title of series 2 is Métaux première transformation and it is a line series.

Example 2: Filling fields with correct values. Currently, iGraph-LITE is able to “guess” the titles of a given graph. It does so quite well. However, computer “guessing” has always been a complex endeavor. As such, it is important that the correct fields are set for the graph (unless of course the titles are somewhat obvious). Notice that you can also move the title box of graphs as if they were regular text boxes. They can also be formatted, and they can be set without showing them.

In particular, it is important to fill the values for the series, even if the legend is not shown. In general, graphing applications such as MS Excel will fill this with bogus titles such as `Series5` and similarly meaningless variables. We have to make sure that the correct titles are provided (see Fig. 1.3 below, where `Series3` should be changed to `IPPI`). As well, make the titles informative, the expression `Seasonally Adjusted` is fine, but not when someone does not know *what* has been seasonally adjusted. Choose something like `Value of Permits (seasonally adjusted)`.

Example 3: Setting the chart type and avoiding bogus data. Graph `c080220a` presents a few abnormalities in its design that makes iGraph-LITE describe it wrongly. Figure 1.4 in page 12 shows the dataset and the properties of the graph in Excel. Several points should be noted: a) Although the graph

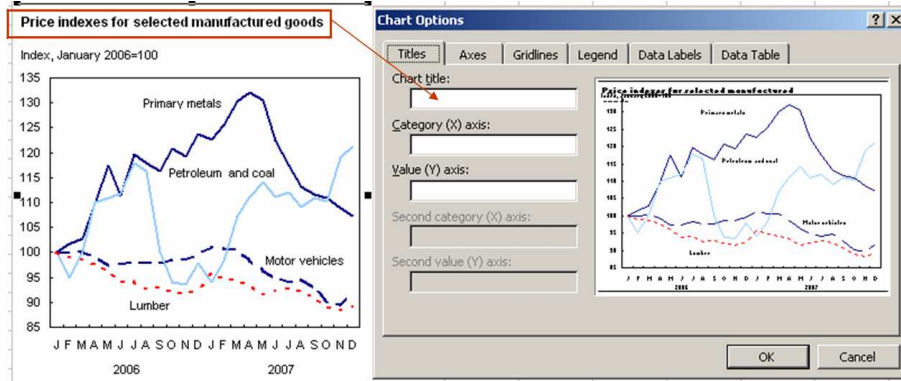


Figure 1.2: Filling the fields for the graph titles

is obviously a bar graph, Excel could only tell that the graph is a combination graph. This is due to the fact that there is a bogus series in it (a series with only zeroes as values which does not seem to fulfill any purpose). We suggest avoiding these kinds of empty series. Notice as well that there is also a bogus value in the primary dataset: it starts at zero, and so iGraph-LITE incorrectly (or, in fact, correctly) assumes that this is the minimum value of the graph. Values should have a meaning. Finally, notice that there is no way to disambiguate the Js in the dataset: they could stand for January, June, or July. Since we have no way of knowing, we will assume that whenever there is ambiguity, the first J stands for January. However, in the best case, they should be identified explicitly, maybe adding some unambiguous category in between, such as O for October.

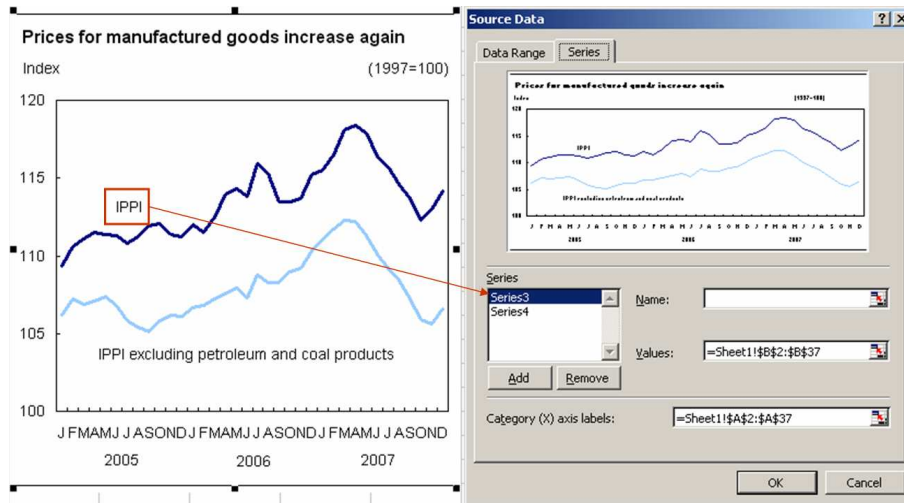


Figure 1.3: Filling the fields for the graph series is IMPORTANT!

1.5. FORMATTING GRAPHS CORRECTLY IN THE GRAPHING APPLICATION11

Example 4: Associate graphs with their data. Graph c081010e could not be processed due to the fact that there is no source data associated to it. The graph actually persisted, but the data did not. All data must accompany the graph, in both languages.

Example 5: Footnotes and complex textual boxes are discouraged. It is quite difficult to find the function of these boxes. Obviously, if they are strictly necessary, they should be included.

Example 6: Never write bogus series. Graph c080220c has a bogus series which is used to represent a straight horizontal black line. Thus, iGraph-LITE reads it as a normal series starting at the beginning of the graph and ending at the end of it, with maximum 0 and minimum 0. Graphical elements should be used for this, such as a straight line with the drawing toolbar.

Example 6: Never write bogus series. Graph has empty text boxes. Even though we clean these boxes as much as possible, iGraph-LITE may attempt to use the information in these boxes to find the titles of the axes, etc. Thus, special care must be exercised when creating textboxes.

INDEX