LogicGem 4.0
User Guide

Contents

Introduction to LogicGem and Decision Tables

LogicGem Overview

LogicGem is a software tool with a two-fold audience, software developers (programmers) and business analysts (non-programmers). The common thread between these two diverse groups is that they both need to design logically complete business rules. For the programmer, the business rules are in fact functional requirements for a software application which is translated into a programming language that becomes part of the application. For the business analyst or domain expert, the same need exists except for translating the logic to a programming language. Business analysts need to develop concise business rules that represent real life or proposed business processes, and these processes need to be every bit as logically robust as a software implementation. Collectively, we can refer to both groups of programmers and business analysts as “logic engineers.”

LogicGem is the appropriate tool for these applications and is based upon a tried and proven methodology called “decision tables.” Decision table technology is the vehicle by which logically complete business rules may be constructed.

The LogicGem logic processor software tool is designed to provide a familiar, easy-to-use way to create, edit, verify and compile decision table logic. LogicGem provides the means to prototype and expand a high-level procedural concept into a decision table, work it through analysis and design phases, and then generate English language documentation and programming source code in various programming languages. LogicGem provides tools to ensure that a table’s logic is complete, unambiguous and contains every applicable rule in an expanded or reduced format.

LogicGem software is a Microsoft Windows™ application to provide a familiar, easy-to-use tool to create, edit, verify and compile decision table logic.

Decision Table Introduction

Decision tables, also known as *logic* tables, have been used for centuries to represent logic in a tabular form. They continue to be a processing method of choice because they provide a quick and easy way to read, understand and execute procedures.

Figures 1 and 2 illustrate how to understand the various components of a decision table.

A decision table is a map representing the relationships of combinations of conditions to combinations of *actions.* The first combination of conditions to actions in Figure 1 is read from top to bottom from the *rule* column as: *if* Condition 1 is true *and* Condition 2 is true *and* Condition 3 is true *then do* Action 1 *and then do* Action 4 *and then do* Action 2.

Condition 1	y	y	y	y	n	n	n	n
Condition 2	y	y	n	n	y	y	n	n
Condition 3	y	n	y	n	y	n	y	n
Action 1	1	2	2		2	1	2	
Action 2	3		3	1		2		2
Action 3					3			
Action 4	2	1	1	2	1	3	1	1

r
u
l
e

Figure 1 – Decision Table Structure

Decision Table Components

The four components of a decision table (*condition stubs, condition entries, action stubs* and *action entries*) are shown in Figure 2.

Conditions to evaluate in a procedure are entered in the *condition stub* area of a decision table. Each condition stub entry is followed with a yes/no matrix in the *condition entry* area of the table. The yes/no matrix is used to represent all the possible true/false combinations of conditions in the table. You can also enter "-" (the hyphen character) in the cell to signify that you "don't care" if the condition's state is true or false.

Actions to perform under a set of true/false conditions are entered in the *action stub* area. A number in the *action entry* area identifies the action(s) to perform and in the required sequence.

The data in the *condition entries* together with the data in the *action entries* is read from top to bottom by column. The data grouped in each column is called a *rule*.

Condition Stubs	Condition Entries
Action Stubs	Action Entries

Figure 2 – Decision Table Components

Decision Table Example

Let's start by taking a look at a simple non-programming example of multiple conditions and actions to determine "What to do today?" Figure 3 depicts one possible decision table to represent this logic.

A single condition/action relationship like "If it is raining then wear your raincoat" is easy to understand and represent in a process but when multiple conditions and actions are involved, the rule-set can get complicated. A decision table helps to simplify and organize logic.

What to do today?								
	1	2	3	4	5	6	7	8
Is today a weekday?	y	y	y	y	n	n	n	n
Is today a holiday?	n	n	y	y	y	y	n	n
Is it raining?	y	n	y	n	y	n	y	n
Go to work	1	1						
Go on a picnic				1		1		1
Watch sports on TV			1		1		1	

Figure 3 – Conditions and Actions mapped into a Decision Table – Expanded View

Figure 3 is an example of a *limited entry table*[1]**.** A limited entry table can only use yes/no condition entries. The condition entry values are "y" for yes and "n" for no. The expanded view means that every possible yes/no combination of conditions is displayed in the condition entries. The combinations of condition and action entries in a column in this example form a *Simple Rule*. The action entry value in this example is "1" to denote the selected action. There are two types of rules: *simple* and *complex*. A simple rule contains either y or n in a condition entry. A complex rule can have a dash in a condition entry. A complex rule is described in the next section, Simplifying Structural Rules.

The example in Figure 3 has eight simple rules because the matrix provides each of the three conditions with two possible values. Therefore the total number of rules in this example is (2*2*2) = 8.

[1] A *limited entry table* uses only y or n (to indicate true or false) in the condition entries. Other types of decision tables are *extended* and *mixed entry tables.* These tables can accommodate more than yes/no condition entries but they are not yet available in LogicGem. Extended and mixed tables will be available in a future release.

Simplifying Structural Rules

The decision table in Figure 3 can be simplified and made smaller without losing any logic content. Certain condition entries can be replaced with dashes to reduce the number of entries. A dash means that the condition entry is ignored – we "don't care" if the value is y or n. Rules with dashes are called *complex rules* because each such rule actually represents more than one simple rule.

For example, the 1st and 2nd rules shown in Figure 3 can be reduced into one complex rule. They can form one complex rule because they have the same action and differ by only one condition entry. Two additional rule pairs in Figure 3 can be reduced into a single complex rule: the 3rd and 5th, and the 4th and 6th rules.

Figure 4 illustrates the example decision table using complex rules. It captures the same logic that is shown in Figure 3 but in a simplified view. In this case, reducing the table eliminated nearly 50% of the rules. The 1st rule is read as follows:

```
IF today is a weekday (yes)  AND
today is a holiday (no)
THEN  go to work.
```

The "Is it raining?" condition is ignored because of the "don't care" dash.

What to do today?								
	1	2	3	4	5	6	7	8
Is today a weekday?	y	-	-	n	n			
Is today a holiday?	n	y	y	n	n			
Is it raining?	-	y	n	y	n			
Go to work	1							
Go on a picnic			1		1			
Watch sports on TV		1		1				

Figure 4 – Condition and Actions mapped into a Decision Table – Structurally Reduced

Simplifying Logic Rules

LogicGem can be used to automatically reduce a decision table's rule-set but it is up to the logic engineer to determine if the rules are logical. LogicGem does not understand the condition and action stubs entered in a

decision table and they do not hold any syntactic or semantic value. Stubs simply help the logic engineer to insert English phrases or programming language constructs into the framework of a decision table. In the case of programming languages, no syntactic checking is performed. That task is left up to the language compiler used to compile LogicGem generated source code.

For example, while building each rule for the decision table in Figure 4 the developer probably would have entered dashes immediately because (logically) certain conditions are easily identified as candidates for using a "don't care" dash. The logic engineer must carefully consider the logic that is put into each rule.

Decision Table Applications

Logic tables are useful during all phases of software development for the simple reason that all phases of software development need correct logic. The only thing that changes is the conceptual level of the conditions and actions given in the tables. The analysis phase will use high level concepts, the design phase will use lower level concepts and the coding phase will have actual computer language code in the table.

Logic tables are easy to read and can convey more information in a smaller space and clearer format than conventional documentation techniques. Logic tables are also easier for both business analysts and users to construct and alter than flowcharts, pseudo-code, narratives and other conventional system analysis techniques. The non-analyst user can be given an empty logic table and be asked to fill it out himself. He can provide information to the business analyst who would normally require intensive interviews and a great deal of time to obtain.

Logic tables can be used to automatically generate source code in any programming language which supports if/then/else or switch/case control constructs. While the time saved by generating code would be worth the effort of learning logic tables, there are other bonuses.

The first bonus is that the code from a perfect logic table will have far fewer bugs and less complexity than a program written by a human programmer. A human being has extreme difficulty in keeping in mind more than six or seven things at the same time. As the number of decisions in a module approaches ten, it is increasingly harder to understand and maintain. This is why most bugs are control flow problems, and not computations.

The second bonus is that LogicGem can optimize the generated code by considering the relative cost and frequency of the whole logic table. The cost of a rule is a number assigned to performing the associated actions of the rule. The cost can be subjective (estimated difficulty on a scale from one to ten) or objective (actual computer time and resource used). For example, “write a letter” would have a lower cost than “write an epic poem” on a subjective scale. The frequency of a rule is how often the rule is expected to be invoked. The frequency can be estimated or actual, and is often expressed as a percentage for convenience. For example, an automobile insurance company might know that 80% of their clients are between the ages of 18 and 25 who have not taken a drivers education course. This frequency information would be used to optimize generated computer source code.

Cost and frequency are independent of each other, and both are independent of the structure of the logic table. LogicGem allows the analyst to change cost and frequency for rules and immediately create completely new optimized source code at the click of a mouse. Furthermore, an option in the generated computer source code will save the frequency information, so that the program can be changed as the environment changes.

Decision Table Terminology

A logic table is a graphic method for representing, testing, and implementing a process. It builds a set of rules for taking actions based on a set of conditions. The table can then be tested for completeness and correctness, and then used to generate optimal source code for computer programs or documentation for humans.

The part of the logic table where the conditions appear is called the condition stub. The part of the logic table where the actions appear is called the action stub. The condition stub appears on the upper left hand side of the logic table and the action stub appears directly below it.

A logic table made up solely of conditions that have binary (or y/n) values is called a *limited entry table*. Limited entry tables are the only kind supported by LogicGem 3.0. Tables having only conditions with greater than binary values are called *extended entry tables*. Logic tables that have both binary and greater than binary conditions are called *mixed entry tables*.

The condition entries appear in columns to the right of the condition stub. These consist of possible values of the condition. Conditions for which any values will do within a rule are read as “don’t care” and are shown with dashes in the condition stub.

Conditions which have some relationship among themselves and eliminate certain combinations of conditions are called a *dependent condition set*. For example, three binary traffic signal conditions (red? (y/n), yellow?

(y/n), green? (y/n)), are a dependent condition set because a traffic signal cannot be red, yellow, and green at the same time.

LogicGem cannot determine if a dependent condition set exists in the condition stub. This requires semantic information about the problem which only a human being can provide.

The action entries appear in columns to the right of the action stub. These are whole numbers which give the order of execution of the actions, or blanks which tell you that an action is not performed as a result of this rule.

In some problems, the order of execution is not important and in others it is vital. For example, in a recipe for a cake you can add flour and sugar to a mixing bowl in either order, but you cannot bake the cake before you have mixed the ingredients together. LogicGem uses ordering in the action stub since unordered actions can be simulated by always numbering the actions from top to bottom in their column.

The condition entry and the matching action entry together are called a rule. Rules can be stated in an if/then form; “If all condition entries are all met, then perform all action entries in the given order.”

The dashes or “don’t care” values in the condition stub of a rule are really a shorthand for a set of rules. A rule without dashes is called a *simple rule*. A rule with dashes is called a *complex rule*.

LogicGem needs to expand this shorthand to perform certain checking operations on the table. Expansion is done by the following procedure:

1. Pick a rule with a dash in at least one row.
2. Replace that rule (column) with two (2) duplicates of itself, since the current version of LogicGem only deals with binary values.
3. Replace the dashes in the selected row of the duplicate columns with all possible values of that condition.
4. Repeat steps 1, 2 and 3 until the logical table has only simple rules.

Complex rules which have no simple rules in common are called disjoint. Complex rules which do have one or more simple rules in common are called overlapping. The overlapping rules can cause two problems, redundancy and contradiction, which are collectively referred to as *ambiguity*.

A *redundancy* means that the same simple rule was hidden in two or more complex rules. This is not fatal but it wastes space, confuses the user, and will generate less than optimal results.

A *contradiction* means that two or more simple rules have the same condition entry, but two or more different action entries. This is serious, and must be removed before any valued results can be gotten from the logic table. LogicGem will assume that different actions with the same conditions exclude each other when it reports possible contradictions. This might not actually be the case, but only the logic engineer can determine this. LogicGem has a choice of strategies for removing contradictions, which will be discussed later.

If a table has as many rules represented as the Completion Ratio says it should, and there is no contradiction or redundancy, the table is said to be *mechanically perfect*. That is, all possible combinations of condition values are accounted for and none can lead to contradictory or redundant action sets. This is the primary goal of LogicGem.

Decision Table Errors

As discussed in the previous section, each rule in a decision table should be different. A decision table is referred to as *mechanically perfect* when all contradictions and redundancies are removed. In this section we'll review a few examples of common errors occurring in decision tables.

Table 1 itemizes the tools available in LogicGem to automatically uncover common decision table errors.

Incompleteness:	Not all conditions are covered.
Contradiction:	Two rules with the same conditions lead to different actions.
Redundancy:	Two rules with the same conditions lead to the same action.
Ambiguity:	A reduced table with contradictory and/or redundancy errors.

Table 1 – Common Decision Table Errors

Figure 5 shows a decision table exhibiting examples of redundant, contradictory and ambiguous rules.

Rule 4 and Rule 5 are redundant since the both contain the same conditions and action.

Rule 4 and Rule 5 are contradictory to Rule 6 since the same conditions map to different actions.

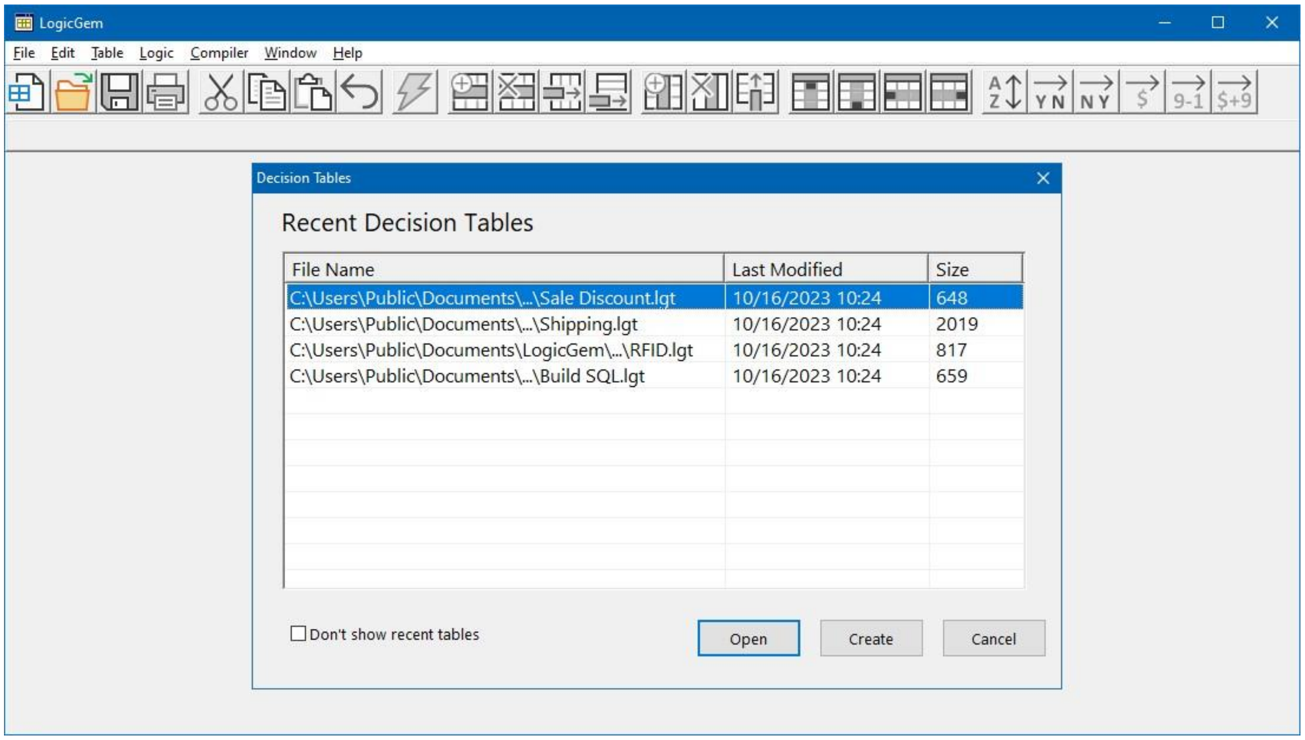
Rule 1 and Rule 7 are ambiguous since their conditions, when considering the "don't cares," overlap with different actions.

What to do today?								
	1	2	3	4	5	6	7	8
Is today a weekday?	y	-	y	n	n	n	-	
Is today a holiday?	n	y	y	y	y	y	n	
Is it raining?	-	y	n	n	n	n	-	
Go to work	1					1		
Go on a picnic			1	1	1			
Watch sports on TV		1					1	

Figure 5 – Common Decision Table Errors

Starting LogicGem

After installing LogicGem on your Windows PC you can launch the program by selecting *LogicGem 3.0* from the LogicGem program group in your Start menu. The following Main Window should be displayed.



File Name	Last Modified	Size
C:\Users\Public\Documents\...\Sale Discount.lgt	10/16/2023 10:24	648
C:\Users\Public\Documents\...\Shipping.lgt	10/16/2023 10:24	2019
C:\Users\Public\Documents\LogicGem\...\RFID.lgt	10/16/2023 10:24	817
C:\Users\Public\Documents\...\Build SQL.lgt	10/16/2023 10:24	659

☐ Don't show recent tables

Figure 6 – LogicGem Main Window

The LogicGem name and logo are identified in the title bar. The menu bar functions are File, Edit, Table, Logic, Compiler, Window and Help.

The File, Edit, Window and Help provide most of the same functions that are found in other applications running under Microsoft Windows. The remaining menu functions that are unique to LogicGem will be explained later using examples that create, edit, verify and compile a sample decision table.

The Toolbar

The tool bar provides quick access to commonly used menu bar functions. Every tool bar function can also be accessed by selecting items from the various menus.

Table 2 contains a complete list of toolbar icons and their respective actions.

ICON	ACTION
	Create a New Decision Table
	Open an Existing Decision Table

	Save an Open Decision Table
	Print the current Decision Table
	Cut Selected Decision Table Information
	Copy Selected Decision Table Information
	Paste Selected Decision Table Information (replaces any existing information)
	Only Condition Entries. Applies edit(s) to the condition entries component only (see the red column area of icon).
	Only Action Entries. Applies edit(s) to the action entries component only (see the red column area of icon).
	Only Stubs. Applies edit(s) to the stub components only (see the red row area of icon).
	Only Entries. Applies edit(s) to the entry components only (see the red row area of icon).
	Undo Last Edit Function. Multiple undo's are available.
	Compiles the current Decision Table using the current compile options, and displays the Compile window.
	Paste Insert Columns. Used with cut or copy columns. Cut or copied columns are inserted into a new selected (highlighted) location. It does not replace the selected column, it is inserted to the left of the selected column.

ICON	ACTION
	Insert Column. An existing column must be selected (highlighted) to use this function. It inserts new empty column to the left of a selected column. May be used repeated times.
	Delete Column. An existing column must be selected (highlighted) to use this function. It deletes the selected column.
	Paste Append Rows. Used with cut or copy rows. Cut or copied rows are appended to the end of either the condition or action rows, whichever is appropriate.
	Paste Insert Rows. Used with cut or copy rows. Cut or copied rows are inserted into a new selected (highlighted) location. It does not replace the selected row, it is inserted above the selected row.
	Insert Rows. An existing row must be selected (highlighted) to activate this function. It inserts new empty row(s) above the selected row.
	Delete Rows. An existing row must be selected (highlighted) to activate this function. It deletes the selected row(s).
A Z ↕	Sort Rows Display the table with the condition stubs rearranged to list entries containing dashes last, in increasing complexity.
→ Y N	Sort Columns Yes Before No. (Display the table to list all the rules (columns) containing yes before those containing no.)
→ N Y	Sort Columns No Before Yes. (Display the table to list all the rules (columns) containing no before those containing yes.)
→ $	Sort Columns Least Costly First. (Rules are sorted according to their increasing value assigned in the Cost option. Blank is the lowest value.)
→ 9-1	Sort Columns Most Frequent First. (Rules are sorted according to their decreasing value assigned in the Frequency option. Blank is the lowest value.)
→ $+9	Sort Columns Cost / Frequency Heuristic. (Rules are sorted according to their decreasing value of combined cost-frequency ratio.)

Table 2 – Toolbar Icons and Actions

Opening, Saving and Printing

Opening a Decision Table

LogicGem can be used to work on new or existing decision tables (a decision table is stored as a file with a .lgt file extension). Decision tables are opened inside of the Worksheet window. The Worksheet window is shown in Figure 7.

Open an Existing Decision Table in one of five different ways:

Drag and drop an existing decision table (.lgt) file into the LogicGem main window.

Double-click on a decision table (.lgt) file in a Windows directory listing to automatically open it in LogicGem.

Click on the File drop-down menu, select a file from the previously opened files provided at the bottom of the menu.

Click on the File drop-down menu, click on the Open option to bring up a selection menu and locate the specific (.lgt) file to open.

Click on the icon to open a file selection dialog and locate the specific (.lgt) file to open.

Create a New Table in one of two ways:

Click on the New option from the File menu.

Click on the icon to create a new Worksheet.

Multiple Worksheets can be opened and accessed from within LogicGem. Use the Window menu to select a table if more than one is open at once. The active table has a check mark before the table name.

Close a Table in one of two ways:

Click on the Close option from the File menu.

Click on the close button of the Worksheet window you wish to close.

In either case, closing a table may result in LogicGem asking “Do you want to save the changes you made to Worksheet?” This prompt appears if you edited your table since opening it, and thus prevents you from losing possibly valuable changes in your table. You can respond by clicking Yes, No, or Cancel your request to close the table.

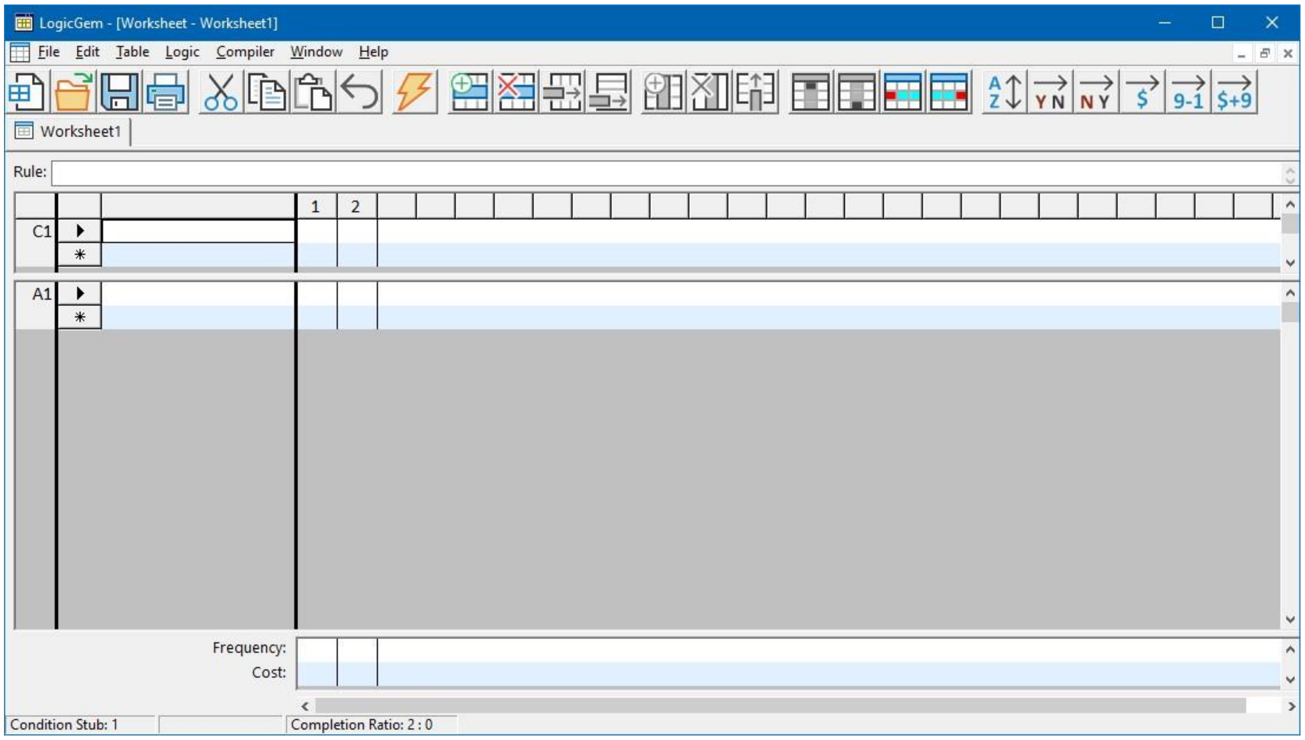

LogicGem - [Worksheet - Worksheet1]
File Edit Table Logic Compiler Window Help
Worksheet1
Rule:
1 2
C1
A1
Frequency:
Cost:
Condition Stub: 1
Completion Ratio: 2 : 0

Figure 7 – New Decision Table – Worksheet Window

Saving a Decision Table

A LogicGem decision table is saved in a manner similar to other Microsoft Windows applications using one of the following techniques:

Menu Bar

Click on the File menu to display the menu item list.

Click on Save from the menu item list to save an existing file or create a new file.

Click on Save As… to create an alternate or new file name.

Tool Bar

Click on the 💾 icon to save an existing or new file.

Preferences

The Save Defaults section on the General tab of the Preferences item found on the Edit menu affects how files are saved in LogicGem. Refer to the General Defaults section in the Preferences chapter for more information.

If you are working with a previously saved table, and you've made some modifications but wish to go back to the previously saved version of the table, you can select the Revert to Last Saved menu item from the File menu.

Printing a Decision Table

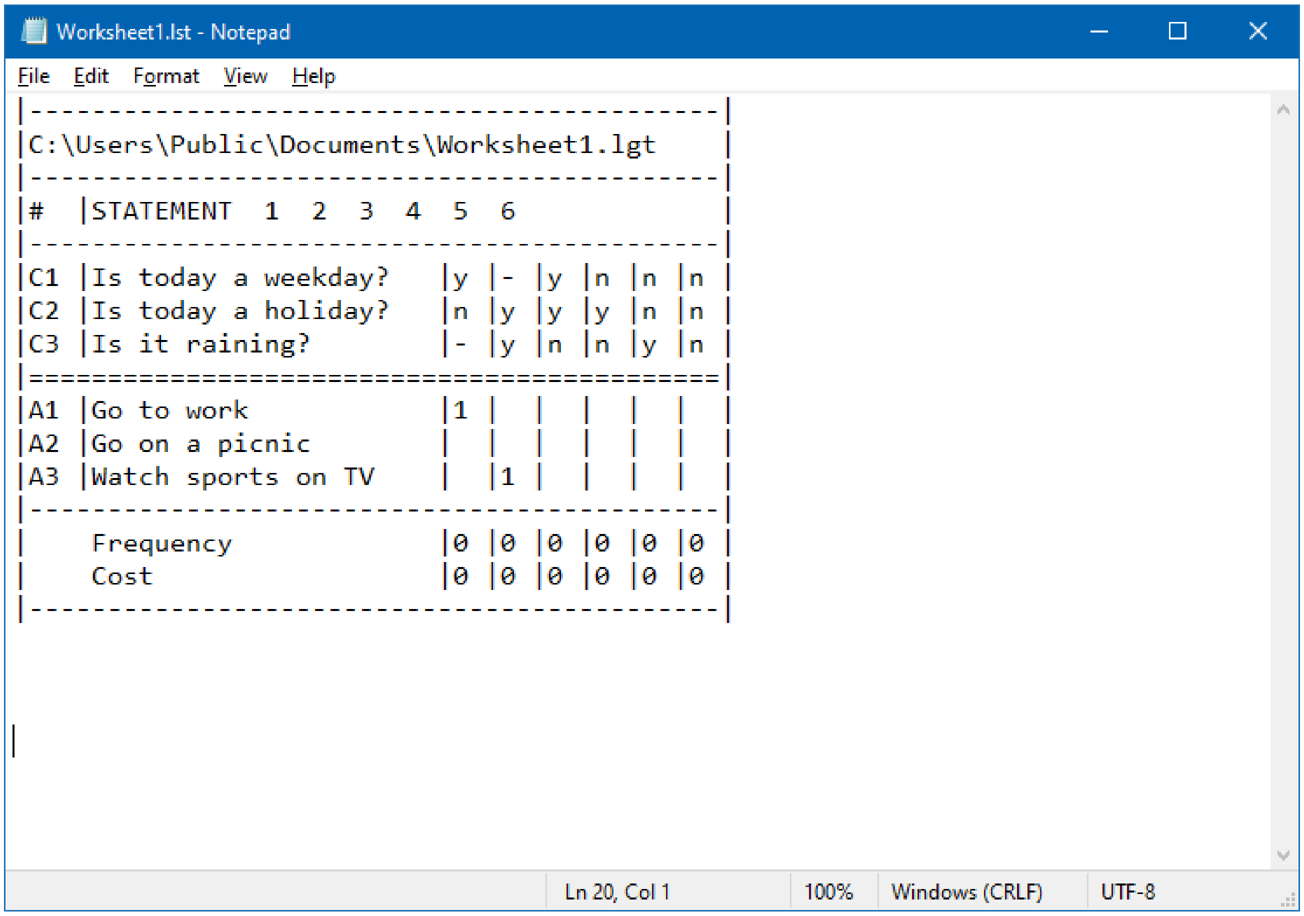
If you wish to print a copy of a decision table, LogicGem offers a simple method for doing so. A hardcopy version of a decision table provides a good way of generating documentation for business logic.

To print:

Click on the File Menu function.

Click on the Print option.

If you have more than one decision table open at the same time, only the active (currently selected) decision table is printed. The printed version of the decision table resembles the Worksheet view in LogicGem. Figure 8 shows an example of a printed decision table.



```
|-------------------------------------------|
|C:\Users\Public\Documents\Worksheet1.lgt    |
|-------------------------------------------|
|#   |STATEMENT  1   2   3   4   5   6       |
|-------------------------------------------|
|C1  |Is today a weekday?   |y  |-  |y  |n  |n  |n  |
|C2  |Is today a holiday?   |n  |y  |y  |y  |n  |n  |
|C3  |Is it raining?        |-  |y  |n  |n  |y  |n  |
|===========================================|
|A1  |Go to work            |1  |   |   |   |   |   |
|A2  |Go on a picnic        |   |   |   |   |   |   |
|A3  |Watch sports on TV    |   |1  |   |   |   |   |
|-------------------------------------------|
|    Frequency              |0  |0  |0  |0  |0  |0  |
|    Cost                   |0  |0  |0  |0  |0  |0  |
|-------------------------------------------|
```

Figure 8 – Printed View of a Decision Table (.LST file)

Saving Output Files

As an alternative to saving a decision table as a LogicGem Table (.lgt), you can also save to two types of output files: Text file (.lst) or HTML (.html).

To Save to an Output File:

Click on the File Menu function.

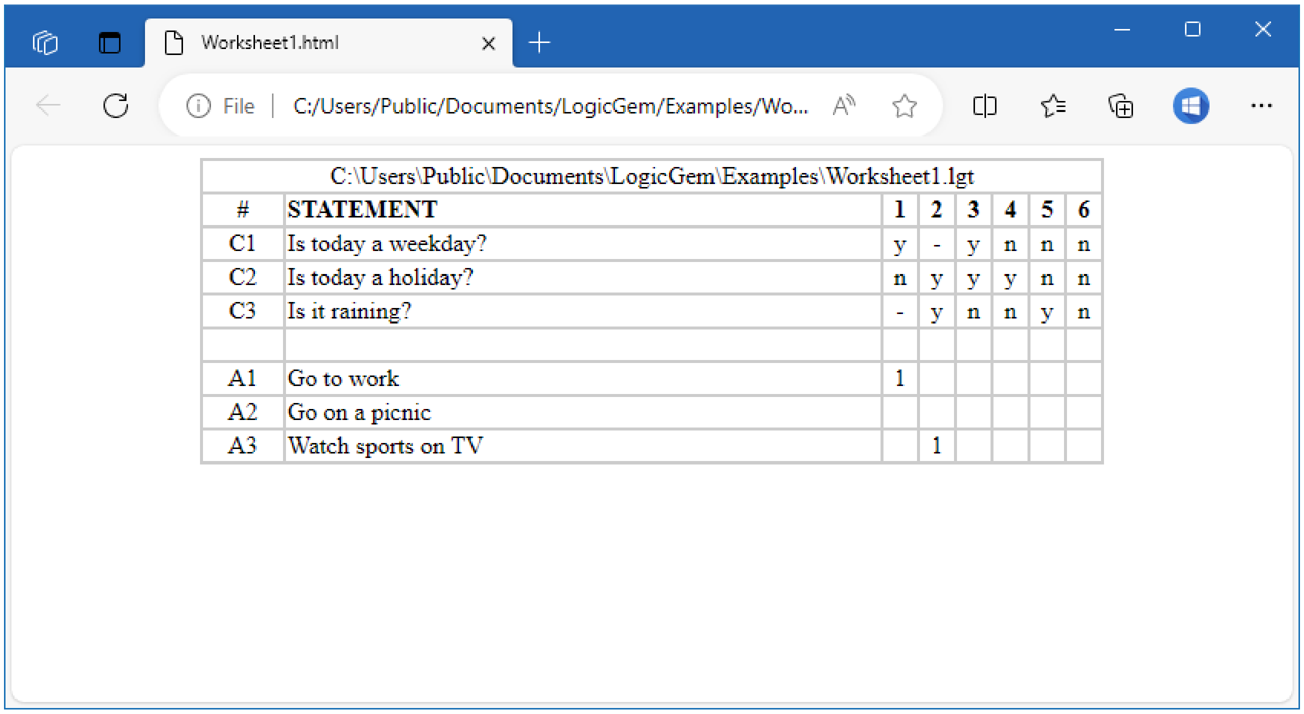
Click on the Output To option.

A window titled "Output To …" is displayed. Select a desired directory and enter a file name to use as the output file.

Click on the Save button.

There are two sections of the Preferences dialog available from the Edit menu that apply to saving logic tables to output files. First, if Link Keywords were specified in the Preferences dialog, then you may have hyperlinks included in the HTML output file that point to other HTML files for related logic tables. In the "Output To …" section of Preferences you may assign file name extensions to both output file formats. The default extension for the LST format is ".lst" and the default for the HTML format is ".html". These defaults may be changed if you wish.

Figure 9 is an example of an HTML output file.



C:\Users\Public\Documents\LogicGem\Examples\Worksheet1.lgt

#	STATEMENT	1	2	3	4	5	6
C1	Is today a weekday?	y	-	y	n	n	n
C2	Is today a holiday?	n	y	y	y	n	n
C3	Is it raining?	-	y	n	n	y	n
A1	Go to work	1					
A2	Go on a picnic						
A3	Watch sports on TV		1				

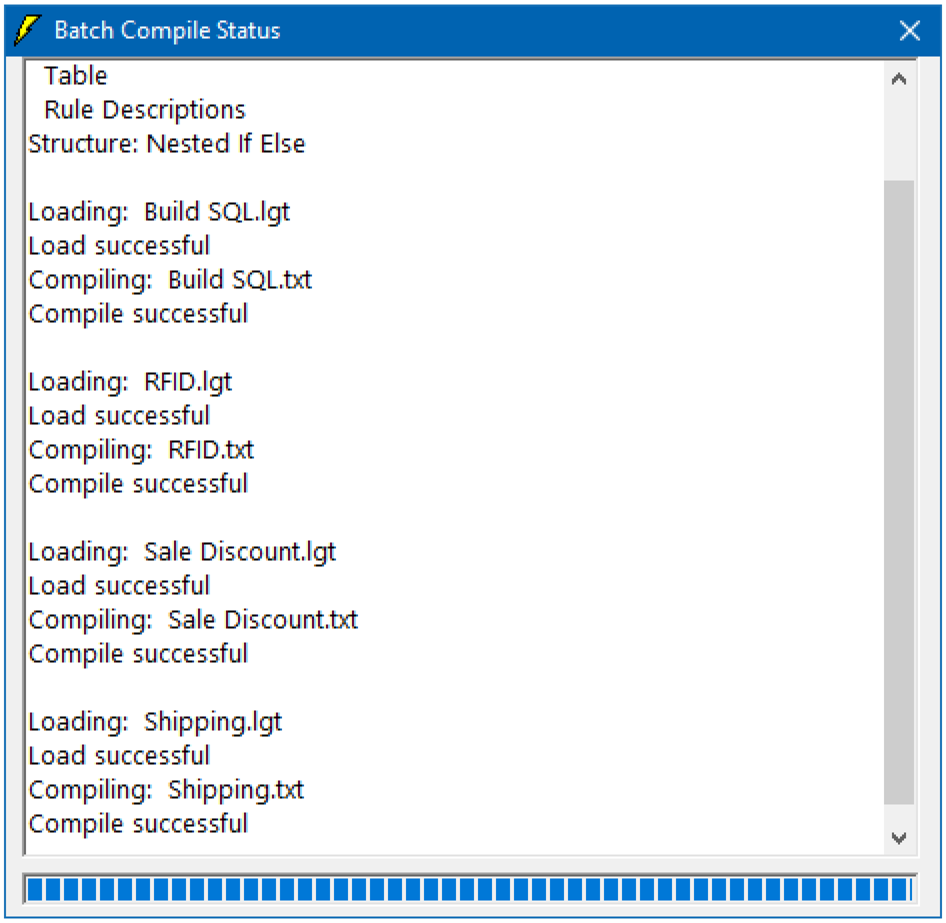
Figure 9 – View of HTML Output File

Batch Operations

The Batch Operations option available on the File menu provides several features offered as a convenience to logic engineers working with a large number of interrelated decision tables. Table 3 describes each function found in the Batch Operations menu.

FUNCTION	ACTION
Output To	This function toggles On or Off the process of automatically generating output files for each logic table during a batch operation. The default setting is Off.
Compile	This function toggles On or Off the process of automatically compiling each logic table during a batch operation. The default setting is On.
Select Files and Execute	Selecting this function from the Batch Operations menu displays a typical Windows file dialog. From here you may select one or more logic tables and click Open. LogicGem then batches the process of loading all of the selected table files, and optionally compiling and generating output files for each table depending on the current settings of the Output To and Compile toggles described above. Figure 10 shows a log window of a batch operation.
Select Entire Directory and Execute	Selecting this function from the Batch Operations menu displays a special browser that shows directories and any logic tables (.lgt files) in each. You can select a directory and LogicGem batch operations will load all of the table files in the directory, and optionally compile and generate output files for each table depending on the current settings of the Output To and Compile toggles.

Table 3 – Batch Operations



```
  Table
  Rule Descriptions
Structure: Nested If Else

Loading:  Build SQL.lgt
Load successful
Compiling:  Build SQL.txt
Compile successful

Loading:  RFID.lgt
Load successful
Compiling:  RFID.txt
Compile successful

Loading:  Sale Discount.lgt
Load successful
Compiling:  Sale Discount.txt
Compile successful

Loading:  Shipping.lgt
Load successful
Compiling:  Shipping.txt
Compile successful
```

Figure 10 – Results of a Batch Operation

Saving a Table-Level Comment

A description (up to 64000 characters) can be recorded that applies to the entire table. This description text is saved with the logic table file and will be present in any of the HTML pages and the LST files that are generated subsequently. In addition, when the **Compiler** | **Options** | **Table** option is turned on, the description text will also be placed in the generated code as a comment when the decision table is compiled.

To enter a Table Comment:

Click on the File Menu function.

Click on the Table Comment option.

A window titled “Table Properties” is displayed. Type in the desired description text, and click the window’s close button.

Decision Table Components

Introduction

This chapter shall more formally develop the integral parts of decision table theory by presenting a concise and detailed description of the primary decision table components.

A decision table is a map representing the relationships of combinations of *conditions* to combinations of *actions*. The following sample problem is mapped into the decision table shown in Figure 11.

Sample Problem: What to do today?

Possible conditions affecting decision:	Is today a weekday? Is today a holiday? Is it raining?
Possible actions to do depending on conditions:	Go to work, Go on a picnic, Watch sports on TV

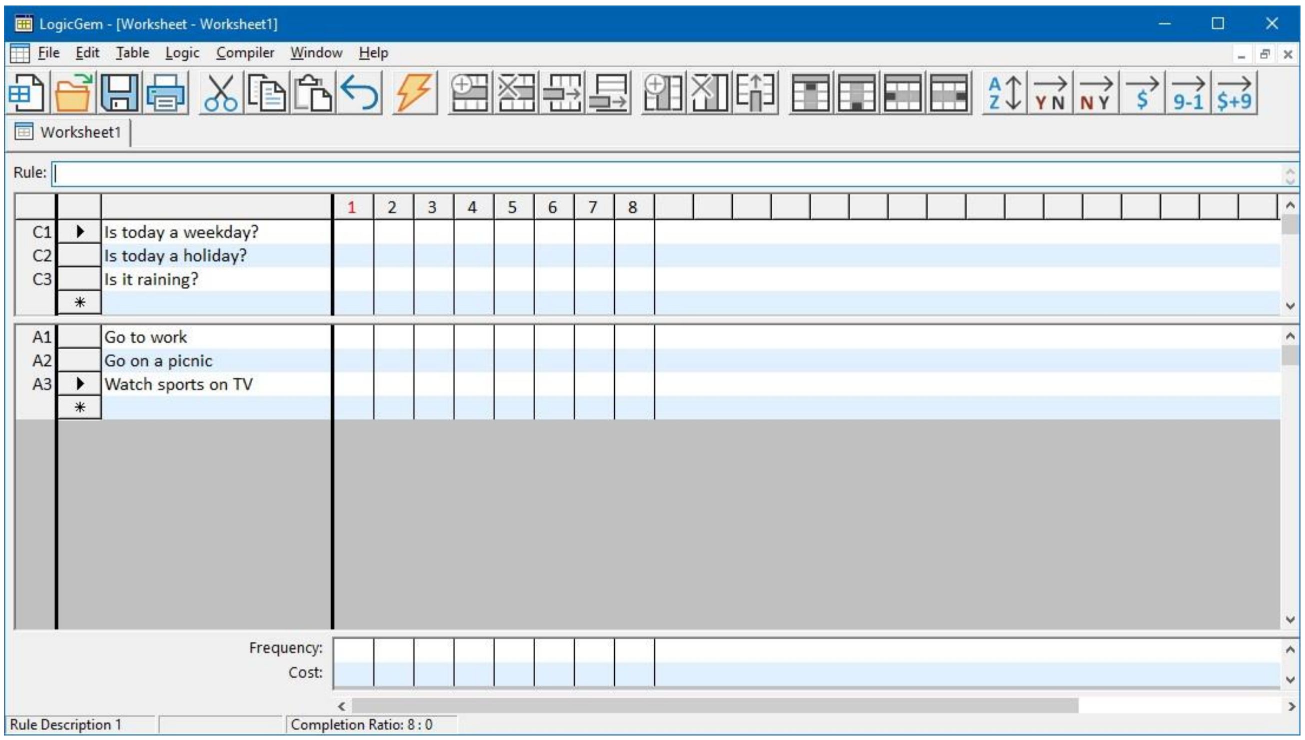
A LogicGem Worksheet and its data can be manipulated like any other Microsoft Windows application.

Worksheet windows can be:

- Minimized, maximized or resized.
- Each component within a Worksheet can be resized. Place the cursor over one of the shaded areas that identify the Conditions, Actions or Columns and the cursor will transform into a set of horizontal or vertical arrows. Drag the line in the direction of the arrows to resize the selected component.

Worksheet data can be added, changed or deleted:

- Click or double-click (depending on the edit) the mouse in the text area and complete the edit or
- Use the Edit drop-down menu, the tool bar icons or the keyboard shortcuts to Cut, Copy or Paste information.


LogicGem - [Worksheet - Worksheet1]
File Edit Table Logic Compiler Window Help
A Z
Y N
N Y
$
9-1
$+9
Worksheet1
Rule:

			1	2	3	4	5	6	7	8
C1	▶	Is today a weekday?								
C2		Is today a holiday?								
C3		Is it raining?								
	*									
A1		Go to work								
A2		Go on a picnic								
A3	▶	Watch sports on TV								
	*									
		Frequency:								
		Cost:								

Rule Description 1 | | Completion Ratio: 8 : 0

Figure 11 – Sample Problem Conditions and Actions mapped into a Decision Table

Three status boxes are displayed at the bottom of each Worksheet. Refer to Figure 11 to see some of the example entries. You can't see all the status boxes entries at once, since status is determined by where the cursor is positioned.

- The first box identifies the active cursor location. Some possible values might be: "Condition Stub: 2" or "Action Entry: 2,3" or "Rule Description 1" (meaning that the cursor is in the Rule Description area for rule 1).
- The second box identifies the number of rules associated with the current cursor position when placed on a rule. A simple rule shows "Column Count: 1", a complex rule would show "Column Count: 2" or greater.
- The third box identifies the *Completion Ratio*. The completion ratio indicates the total number of rules that must be accounted for based on the number of conditions specified, versus the number of rules currently accounted for. For example, a Completion Ratio: 8:6 indicates that you have three binary conditions (2*2*2 = 8) which require 8 rules, and that only 6 rules are currently accounted for.

Condition Stubs

Conditions are the primary ingredient of a decision table as they represent the conditions to evaluate for logical completeness. Conditions are entered in the *Condition Stub* area of a decision table. When LogicGem is used as a programmer's tool, these stubs are presumed to contain syntactically meaningful target source language expressions. The result of a LogicGem compile operation is then compilable by a destination language compiler. Pseudo code or English language type of expressions may also be used here, if you choose to use LogicGem as a business rule tool.

A maximum of 15 condition stubs can be entered in one decision table. Each condition stub entry can be up to 1,024 characters in length. The following process shows how to enter condition stubs:

1. Click the mouse in the small empty area to the right of the *Condition Label* C1, and an arrowhead points to the area.
2. Enter in the first condition and a pencil icon will replace the arrowhead.
3. Click the mouse in the next blank condition space beside the asterisk, to enter the next condition. As each condition is committed it is labeled in ascending sequence as C1, C2, C3, … , C15.
4. Repeat step 3 to enter each condition.

The number of condition stubs determines the number of rules (columns) that must be accounted for during the decision table creation process. The more conditions, also results in a higher Completion Ratio value. Since each condition represents a binary value, the number of rules to be accounted for is always a power of 2 (2^n where n is the number of conditions). For example, a decision table with three conditions requires you to account for 8 rules (2^3), and a table with 15 conditions requires you to consider 32,768 rules (2^{15}).

Action Stubs

Actions are another important component of a decision table as they represent actions to perform under a set of conditions. Actions are entered in the *Action Stub* area of a decision table. When LogicGem is used as a programmer's tool, these stubs are presumed to contain syntactically correct target source language statements. Pseudo code or English language type of statements may also be used here if you choose to use LogicGem as a business rule tool.

Up to 32,767 actions stubs can be entered in one decision table. Each action stub can be up to 1,024 characters in length. The following process shows how to enter action stubs:

1. Click the mouse in the small empty area to the right of the *Action Label* A1, and an arrowhead points to the area.
2. Enter in the first action and a pencil icon will replace the arrowhead.
3. Click the mouse in the next blank action space beside the asterisk, to enter the next action. As each action is committed it is labeled in ascending sequence as A1, A2, A3 and so on.
4. Repeat step 3 to enter each action.

Table Links

In either Condition Stubs or Action Stubs, you can enter a link keyword followed by an existing decision table name. The purpose of *Table Links* is simply as a convenience for opening decision tables. Clicking on a table link is therefore a context driven shortcut for using the **File** | **Open** option. The specific choice for link keywords is entirely up to you. Once you've defined your link keywords, LogicGem simply recognizes them in the stub areas, and provides a hyperlink to the referenced decision table.

You are able to define your link keywords in the Preferences dialog. Select the **Edit** | **Preferences** option and enter the link keywords in the Link Keywords field on the General tab. For instance, you can enter "goto" and "link" as link keywords. Multiple link keywords may be specified. Now you can enter a condition stub such as:

```
goto table.lgt
```

In the LogicGem worksheet, the stubs using link keywords will be shown as a link by being underlined and in blue text (much like a hyperlink), and if you click on this stub, table.lgt will be opened for you within LogicGem. Also, if you output to HTML, these links will be active in HTML format as well.

In order to edit or select an existing table link, you can right-click to get the normal behavior a left-click normally provides. Once in edit mode, then left clicks behave as usual.

Links can either include the .lgt extension or not. If a path to the decision table is not supplied in the stub, then the directory location of the current decision table is used. This means no path is needed if linking decision tables exist in the same directory. Alternately, you can enter a static path which will be used when looking for the decision table to open.

As an example, let's assume that "goto" and "link" are both previously entered as Link Keywords in Preferences, then all of the following stubs are allowed.

"goto table1.lgt"

"link table2"

"link c:\dtables\table3.lgt"

Condition Entries

The *Condition Entry* area of a decision table is where you represent the possible true/false combinations of conditions that constitute a rule. LogicGem accepts three values for condition entries: "y" for Yes/True values, "n" for No/False values, and the "-" dash, meaning "don't care".

NOTE: Condition entries can be in upper or lowercase, but they are interpreted differently:

Lowercase condition entries are interpreted as y = yes and n = no.

Uppercase condition entries are interpreted as Y = no and N = yes.

The reason for including the "not equal to" equivalents is that it provides one more convenience for representing logic in a concise manner. To avoid confusion and improve readability of your decision tables, lowercase values should normally be used throughout and set as the default option in the Ignore Case on Entries setting found in the General tab in the Preferences option of the Edit menu.

The following process shows how to enter condition entries:

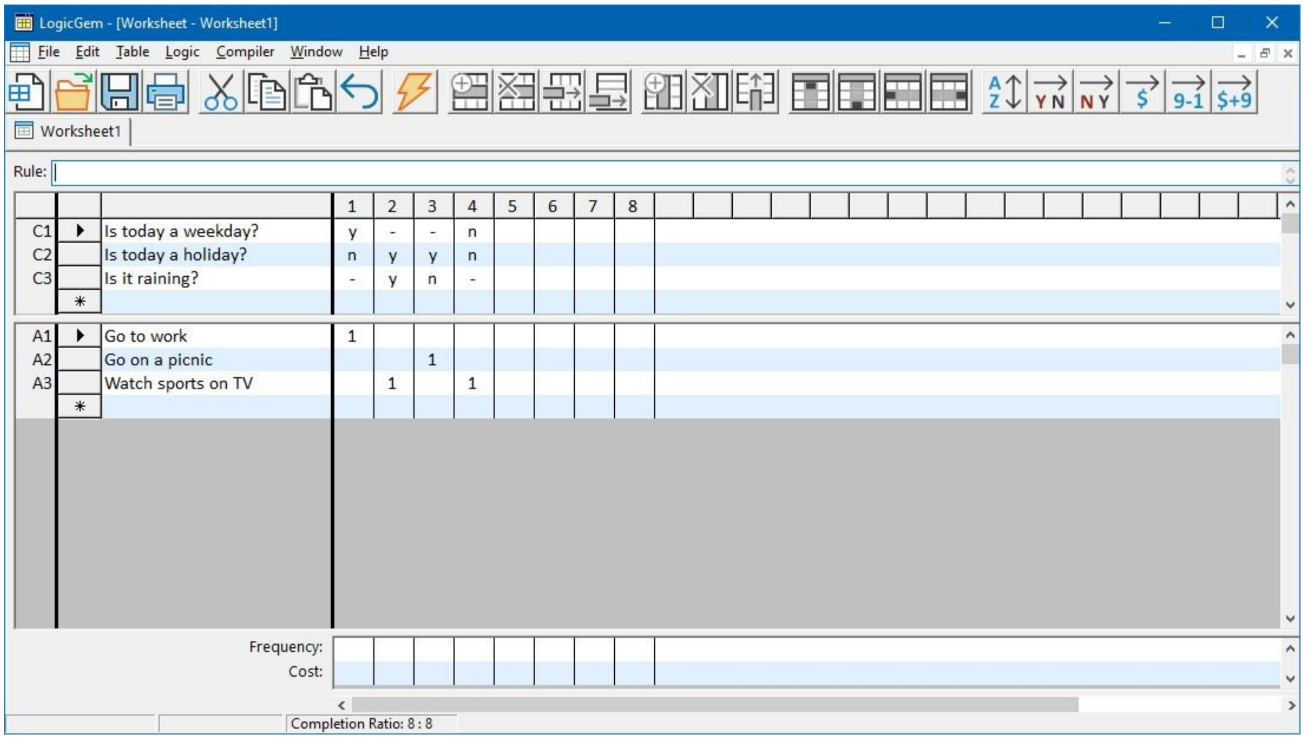
1. Click on a condition column and enter either a **y** or **n** in the box provided or
2. Select the Expand option from the Logic drop-down menu. Every possible yes/no combination will be entered automatically into the condition entry area.

Action Entries

The *Action Entry* area of a decision table identifies the action(s) to perform under a set of true/false conditions. More specifically, the actions specified in this area define what must occur when a rule is found to apply. You enter single digit values (1-9) in the action entry area to indicate the order of execution of the actions in that column. A single action to perform under one set of conditions (rule) is identified by the number 1 for example. If multiple actions are to be performed for one set of conditions (rule) they can be numbered according to the required execution sequence or they can be assigned the same number and executed top to bottom. If there are more than nine actions under a rule, you can number all the actions greater than nine with the number 9, and they will be executed in top to bottom order.

For example, the first rule in Figure 12 could have also included an action entry of 1 in Action 3 "watch sports on TV." Rule 1 would be read as "Go to work **and** Watch sports on TV." But for Rule 2, an additional action entry of 2 in Action 2 would be read as "Watch sports on TV **then** Go on a picnic."

To enter action entries, click on an action column and enter the appropriate number in the box provided.


LogicGem - [Worksheet - Worksheet1]
File Edit Table Logic Compiler Window Help
Worksheet1
Rule:
Frequency:
Cost:
Completion Ratio: 8 : 8

		1	2	3	4	5	6	7	8
C1	Is today a weekday?	y	-	-	n				
C2	Is today a holiday?	n	y	y	n				
C3	Is it raining?	-	y	n	-				
A1	Go to work	1							
A2	Go on a picnic			1					
A3	Watch sports on TV		1		1				

Figure 12 – Condition and Action Entries Added to Example Decision Table

Rule Descriptions

The data in the *Condition Entries* together with the data in the A*ction Entries* is read from top to bottom by column. The data grouped in each column is called a *Rule*. Optionally, a rule can be given a description before or after data are entered. Adding a rule description is recommended as documentation for a complex set of conditions and/or a lengthy series of actions. Not all rules need be documented in this way, only the complex rules should have descriptions.

To enter a rule description, follow the process below:

1. Click the mouse anywhere in the rule column to be described.
2. Click the mouse in the text box beside the Rule label.
3. Enter the rule description text up to a maximum of 1,024 characters.

Frequency

The Frequency row appears under the Action section. The logic engineer can optionally assign for each rule an estimate of how often it might be invoked. A frequency number can be up to three digits. Very often, the frequency number is treated as a percentage although LogicGem does not check if all the frequency numbers add up to 100%. Obviously, the frequency number should be greater than zero, otherwise the rule is useless.

The frequency row entries enable the LogicGem compiler to do some optimizations relative to execution speed of the code structure produced.

To enter a frequency numbers for rules, follow the process below:

1. Click the mouse in the frequency row under a desired rule.
2. Enter a frequency number in the box provided.
3. Repeat steps 1 and 2 to enter each frequency as desired (not all rules are required to have a frequency number).

Cost

The Cost row appears under the Action section. A rule can optionally be assigned an estimate of how much it costs to be executed. A Cost can be up to three digits and use any unit of measure. Examples are actual monetary cost, amount of computer time, storage space, or any measure that can be applied to all of the rules in the logic table (i.e. do not put CPU seconds in one column and dollars in another). These cost numbers come into play at compile time. The compiler uses the cost entry to optimize the generated program code.

To enter a cost numbers for rules, follow the process below:

1. Click the mouse in the cost row under a desired rule.
2. Enter a cost number in the box provided.
3. Repeat steps 1 and 2 to enter each cost (not all rules are required to have a cost number).

The Frequency and Cost values are not considered as exact amounts. Values assigned to these entries are used to establish a relative relationship among the rules, such as “Rule 3 is more frequent (or costly) than Rule 6” and “Rule 2 is more frequent than Rule 7,” and so forth. Thus, it is important that we know only a rough approximation of Frequency or Cost, which after all, is the only type you are likely to have available in most instances.

Edit and Sort Functions

Introduction

In addition to the regular Microsoft Windows edit functions (cut, copy, paste, find and replace) LogicGem offers special edit functions available to assist with editing and sorting decision table components.

The special edit and sort functions are accessed from the Edit menu, Table menu, Tool Bar or Preferences dialog box. Refer back to the Toolbar section in the 'Introduction to LogicGem 3.0 and Decision Tables' chapter for a description of each special function.

This chapter provides a description of the edit and sort functions, and the various ways to use them. It may be useful for you to create the table shown in Figure 12 and try some of the edit functions.

Edit Functions by Row

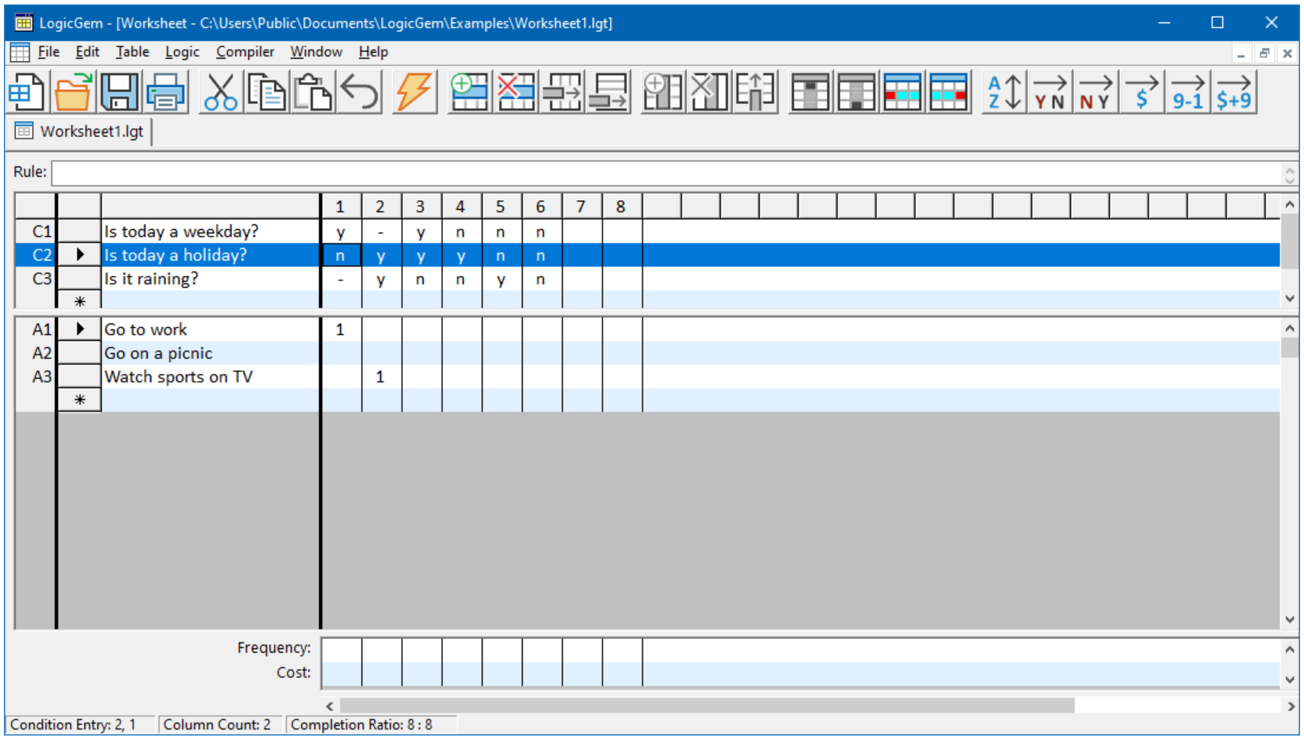
The following edit functions are available when a row is selected from the decision table. For example, the row tool bar icons are highlighted in Figure 13 because Condition 2 is selected. The functions may operate on multiple rows. To select more than one row, use the Shift and Control keys in usual way for Windows.

Table 4 below shows the functions available from the tool bar or the Edit and Table menus.

FUNCTION	ACTION
Cut Rows	The selected row(s) are cut from the decision table and copied to the clipboard. This function is available in the Edit menu only when a row is selected in the table.
Copy Rows	The selected row(s) are copied from the decision table and copied to the clipboard. This function is available in the Edit menu only when a row is selected in the table.
Paste Rows	The row(s) in the clipboard are pasted into the table at the selected location. This function is available in the Edit menu only when a row is selected in the table.
Paste / Insert Rows	Used with cut or copy rows functions. Cut or copied rows are inserted into a new selected (highlighted) location. It does not replace the selected row, it is insert above the selected row. This function is available in the Edit menu only when a row is selected in the table. This function is also available as a tool bar icon.

FUNCTION	ACTION
Paste / Append Rows	Used with cut or copy rows functions. Cut or copied rows are appended to the end of selected section of the table, either Conditions or Actions. If a row was previously copied or cut to the clipboard, the new row will contain this content; otherwise an empty row is appended. This function is available in the Edit menu only when a row is selected in the table. This function is also available as a tool bar icon.
Row / Insert	A new empty row is inserted into the decision table above the selected row. This function is available in the Table menu only when a row is selected in the table. At all other times, this function is grayed out. This function is also available as a tool bar icon (Insert Rows).
Row / Delete	The selected row(s) are deleted from the decision table. This function is available in the Table menu only when a row is selected in the table. At all other times, this function is grayed out. This function is also available as a toolbar icon (Delete Rows).

Table 4 – Row Edit Functions


LogicGem - [Worksheet - C:\Users\Public\Documents\LogicGem\Examples\Worksheet1.lgt]
File Edit Table Logic Compiler Window Help
Worksheet1.lgt
Rule:

		1	2	3	4	5	6	7	8
C1	Is today a weekday?	y	-	y	n	n	n		
C2	Is today a holiday?	n	y	y	y	n	n		
C3	Is it raining?	-	y	n	n	y	n		
A1	Go to work	1							
A2	Go on a picnic								
A3	Watch sports on TV		1						
	Frequency:								
	Cost:								

Condition Entry: 2, 1 | Column Count: 2 | Completion Ratio: 8 : 8

Figure 13 – Edit by Row

Edit by Row Component

Each of the row edit functions can optionally be applied to a complete row, only the Stub or just the Entry component. To limit an edit to a portion of a table row, the component must be selected from the Edit menu options, toolbar or Preferences dialog box **before** applying the edit function.

To limit an edit to a specific row component, highlight the row and use one of the following techniques:

Edit Menu

A checkmark beside either the **Only Entries** or **Only Stubs** option limits the editing function to the selected component. Click on the option to select or de-select it.

Tool Bar

Click on either the Only Stubs icon or the Only Entries icon to restrict the edit.

Preferences

Settings made in the Preferences dialog box will remain in effect from session to session.

Click on the Edit menu function then click on the Preferences option.

The Preferences dialog box provides tabs to select and display the General, Compiler, or Appearance options.

Click on the General tab or use the left/right arrow keys to move between tabs.

In the Edit Behavior section of the dialog box, click on the **Only Stubs** or **Only Entries** checkbox to select or de-select a component.

Edit Functions by Column

The following edit functions are available when a column is selected from the decision table. For example, the column tool bar icons are highlighted in Figure 14 because Rule 2 is selected.

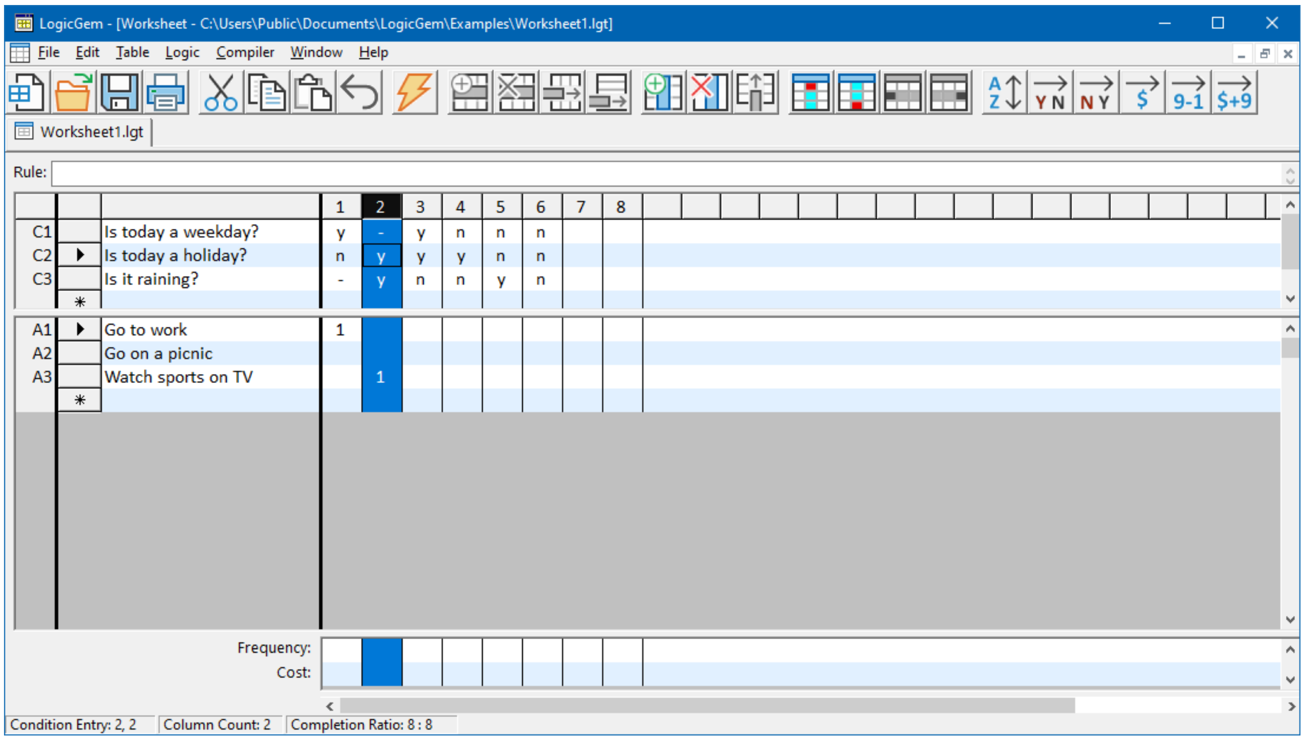
Table 5 below shows the functions available from the tool bar or the Edit and Table menus.

FUNCTION	ACTION
Cut Columns	The selected columns(s) are cut from the decision table and copied to the clipboard. This function is available in the Edit menu only when a column is selected in the table.
Copy Columns	The selected column(s) are copied from the decision table and copied to the clipboard. This function is available in the Edit menu only when a column is selected in the table.
Paste Columns	The column(s) in the clipboard are pasted into the table at the selected location. This function is available in the Edit menu only when a column is selected in the table.

FUNCTION	ACTION
Paste / Insert Columns	Used with cut or copy columns functions. Cut or copied columns are inserted into a new selected (highlighted) location. It does not replace the selected column; it is inserted to the left of the selected column. This function is available in the Edit menu only when a column is selected in the

	table. This function is also available as a tool bar icon.
Column / Insert	A new empty column is inserted into the decision table to the left of the selected column. This function is available in the Table menu only when a column is selected in the table. At all other times, this function is grayed out. This function is also available as a tool bar icon (Insert Columns).
Column / Delete	The selected column(s) are deleted from the decision table. This function is available in the Table menu only when a column is selected in the table. At all other times, this function is grayed out. This function is also available as a toolbar icon (Delete Columns).

Table 5 – Column Edit Functions


LogicGem - [Worksheet - C:\Users\Public\Documents\LogicGem\Examples\Worksheet1.lgt]
File Edit Table Logic Compiler Window Help
Worksheet1.lgt
Rule:

		1	2	3	4	5	6	7	8
C1	Is today a weekday?	y	-	y	n	n	n		
C2	Is today a holiday?	n	y	y	y	n	n		
C3	Is it raining?	-	y	n	n	y	n		
A1	Go to work	1							
A2	Go on a picnic								
A3	Watch sports on TV		1						
	Frequency:								
	Cost:								

Condition Entry: 2, 2 | Column Count: 2 | Completion Ratio: 8 : 8

Figure 14 – Edit by Column

Edit by Column Component

Each of the edit functions can optionally be applied to a complete column, only the Condition Entry or just the Action Entry. To limit an edit, a table component must be selected from the edit menu options, toolbar or Preferences dialog box **before** applying the edit function.

To limit an edit to a specific Column component, highlight the column and use one of the following techniques:

Edit Menu

A checkmark beside either the **Only Condition Rules** or **Only Action Rules** option limits the editing function to the selected component. Click on the option to select or de-select it.

Tool Bar

Click on either the Only Condition Rules icon or the Only Action Rules icon to limit the edit.

Preferences

Settings made in the Preferences dialog box will remain in effect from session to session.

Click on the Edit menu function then click on the Preferences option.

The Preferences dialog box provides tabs to select and display the General, Compiler, or Appearance options.

Click on the General tab or use the left/right arrow keys to move between tabs.

In the Edit Behavior section of the dialog box, click on the **Only Condition Rules** or **Only Action Rules** checkbox to select or de-select a component.

Sort Functions

LogicGem offers several useful Sort functions designed to rearrange rows and/or columns according to six predetermined criteria:

- Sort rows most complex last - Display the table with the condition stubs rearranged to list entries containing dashes last – in increasing complexity.
- Sort columns Yes before No - Display the table to list all the rules (columns) containing Yes before those containing No.
- Sort columns No before Yes - Display the table to list all the rules (columns) containing No before those containing Yes.
- Sort columns least costly first - Rules are sorted according to their increasing value assigned in the Cost option. An empty cost cell is the lowest value.
- Sort columns most frequent first - Rules are sorted according to their decreasing value assigned in the Frequency option. An empty frequency cell is the lowest value.
- Sort columns cost / frequency heuristic - Rules are sorted according to their decreasing value of combined cost-frequency ratio.

You may use one of the following methods to invoke a sort function:

Menu bar

Click on the Table menu function

Click on the Sort by Row option to have the condition rows listed in the table from least to most complex.

Click on the Sort by Column option to select from a sub-menu of sort options. Click on the desired sub-menu option.

Tool Bar

Refer back to the Toolbar section in the ‘Introduction to LogicGem 3.0 and Decision Tables’ chapter for a description of the six different sort icons.

Find and Replace

When building and maintaining large decision tables it is useful to be able to find and optionally replace keywords inside of condition and action stubs as well as rule descriptions. LogicGem offers Find / Replace dialog found in the Edit menu for this purpose (see Figure 15).

The following options are available from the Find / Replace dialog:

- Find What: Enter the desired text to search for and click the Find Next Button. Repeatedly click the Find Next button to find more occurrences.
- Replace With: Enter the desired replacement text and click on the Replace or Replace All button and the found text will be replaced with the replacement text.
- Search - Current Table or All Tables: Limit the search to the current table or search all open tables.
- Match Case: Check the box to match the case exactly as entered or ignore case.
- Include Rule Descriptions: Check the box to search the rule descriptions in the search for the desired text.
- Found Text (at the bottom of the dialog): The Find / Replace function displays the search results here, along with an indication of where in the decision table the text was found (e.g. Action Stub 3). For additional convenience, you can replace or edit the text yourself, and then click the Replace button.
- Previously used find and replace keywords appear in the pull-down lists after subsequent use.

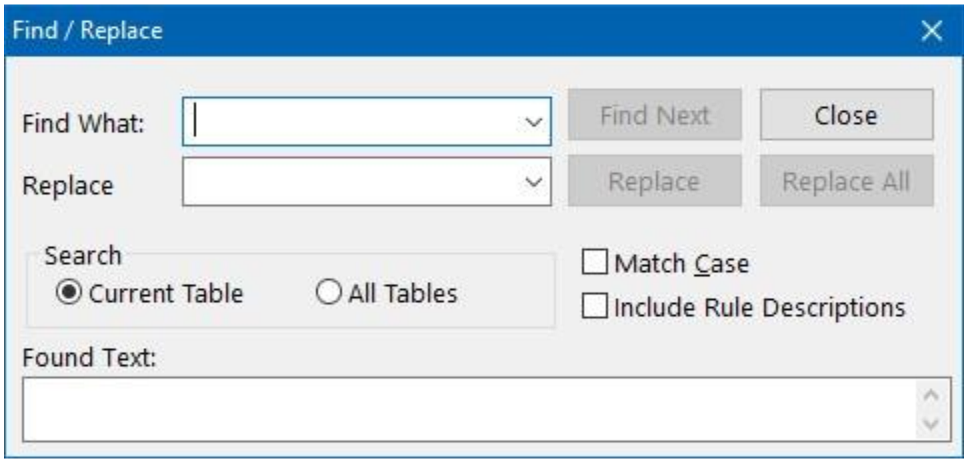


Figure 15 – Find/Replace Dialog

Verify Decision Table Logic

Introduction

The logic processing functions available from the Logic menu are the components that make LogicGem unique in the professional tools arena for software developers. These functions provide the means to verify the structural logic of a decision table. These functions identify and correct rules that are redundant, contradictory or ambiguous. Furthermore, these functions serve to create, analyze and massage a logic table to perfection. Table 6 below summarizes the logic functions which are found in the Logic menu and can be applied to any active Worksheet.

FUNCTION	ACTION
Ambiguity Check	A logic engineering tool used to determine whether a reduced table has contradictory and/or redundancy errors.
Disambiguate	A logic engineering tool used to determine whether a table has ambiguous rules and then proceeds to disambiguate the rules.
Expand	A logic engineering tool used to create the simple rules to represent all possible true/false combinations of conditions in the table.
Missing	A logic engineering tool used to indicate whether any rules are missing and adds the rules to complete the table.
…One Rule at a Time	Similar to the Missing logic engineering tool, but enters only one rule at a time if one or more are missing.
Reduce	A logic engineering tool used to reduce two or more simple or complex rules into one complex rule when certain criteria are met.

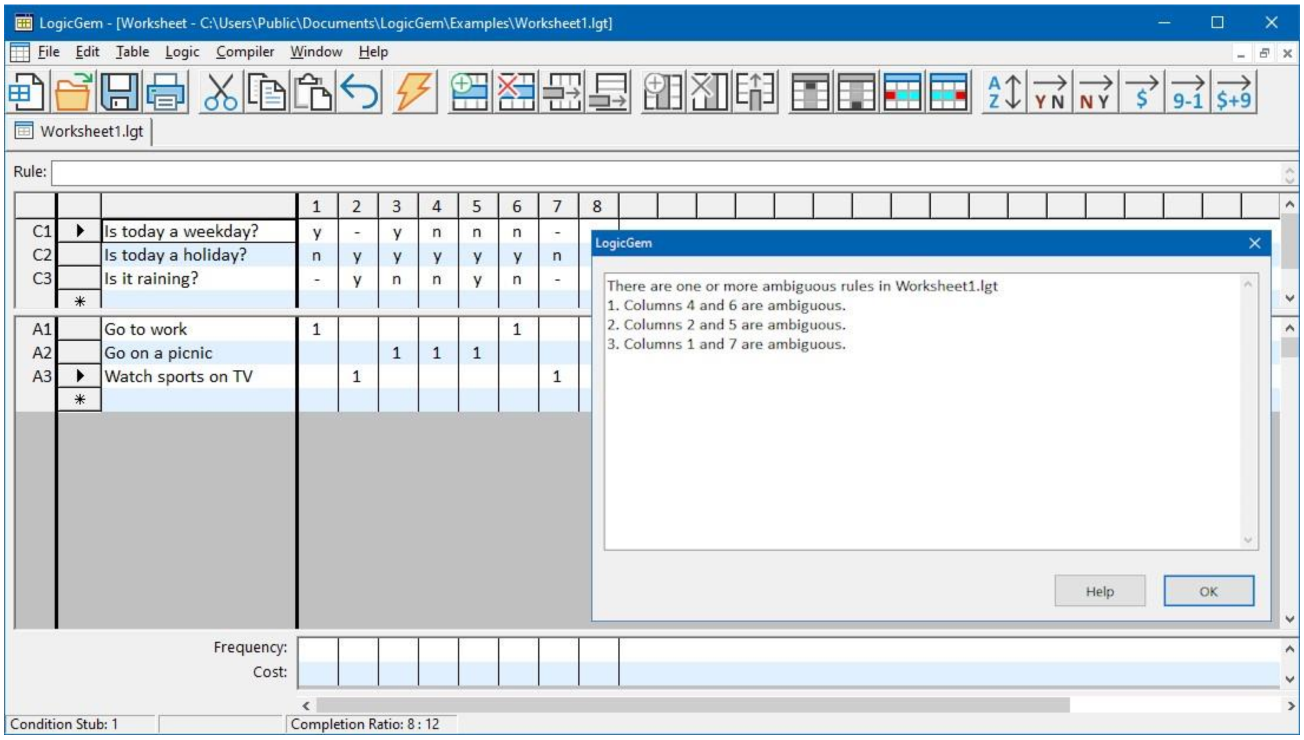
Table 6 – Logic Functions

Ambiguity Check

Ambiguity Check is a logic engineering tool used to determine whether a reduced table has contradictory and/or redundancy errors. Ambiguity Check identifies pair wise all the ambiguous rules in a faulty logic table. It is worth to note here the precise meaning of logic ambiguity. Two complex rules are ambiguous if the two sets of simple rules into which they can be decomposed share at least one common element. Consider the following two cases:

CASE 1:	Two simple rules are ambiguous (redundant) if they are copies of each other.
CASE 2:	Two simple rules are ambiguous (contradictory) if their condition entries are copies of each other, but their action entries are different.

Figure 16 shows an example of a logic table with a number of logic errors. Using the Ambiguity Check tool, we see that rules 5 and 6 were found to be ambiguous. In this case, Case 2 above applies, and the rules are contradictory since C1, C2, and C3 for Rules 5 and 6 are the same, but Rule 5's action is A2 while Rule 6's action is A1.


LogicGem - [Worksheet - C:\Users\Public\Documents\LogicGem\Examples\Worksheet1.lgt]
File Edit Table Logic Compiler Window Help
Worksheet1.lgt
Rule:
Frequency:
Cost:
Condition Stub: 1
Completion Ratio: 8 : 12
LogicGem
Help
OK

		1	2	3	4	5	6	7	8
C1	Is today a weekday?	y	-	y	n	n	n	-	
C2	Is today a holiday?	n	y	y	y	y	y	n	
C3	Is it raining?	-	y	n	n	y	n	-	
A1	Go to work	1					1		
A2	Go on a picnic			1	1	1			
A3	Watch sports on TV		1					1	

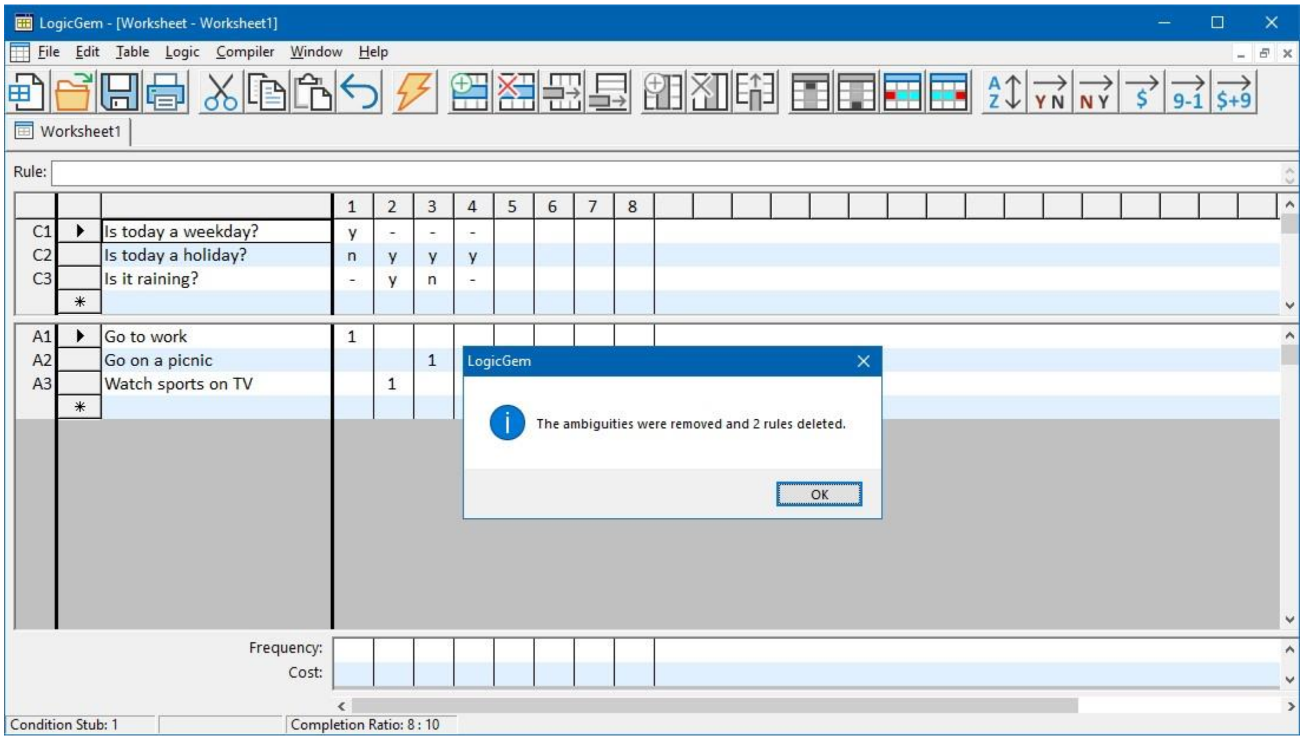
```
There are one or more ambiguous rules in Worksheet1.lgt
1. Columns 4 and 6 are ambiguous.
2. Columns 2 and 5 are ambiguous.
3. Columns 1 and 7 are ambiguous.
```

Figure 16 – Detecting Ambiguous Rules

Disambiguate

Disambiguate is a logic engineering tool used to determine whether a table has ambiguous rules and then proceeds to disambiguate the rules. A pair of ambiguous rules is disambiguated into one unique rule by removing the rule with the lower column count number. The column count of a complex rule is by definition the number of simple rules that it contains. By throwing away the rule with the lower count the Disambiguate function tries to retain as much information as possible about the original table. Sometimes the lower count rule in an ambiguous pair is completely contained in the other higher-count rule. In this case, no information is lost by deleting the former.

In order to demonstrate a point, let's consider our sample logic table with Rule 4's C1 set to dash instead of "n" as we've seen before. Figure 17 shows this table which now has ambiguous rules as result of our change. Notice that the Completion Ratio is 8:10, indicating that too many rules have been accounted for. We can use the Disambiguate function to address the logic problem. The tool correctly finds the two ambiguous rules: Rule 1 and Rule 4 and applies the column count rule. The column count for Rule 1 is 2 and for Rule 4 the column count is 4. This means LogicGem will throw out Rule 1. The process yields a logically complete decision table, but unfortunately action A1 is never referenced. In such cases, the logic engineer must go back to analyze the table to make sure all conditions and actions are being considered properly. The Disambiguate function, in this case, allows us to take a step in the right direction.


LogicGem - [Worksheet - Worksheet1]
File Edit Table Logic Compiler Window Help
Worksheet1
Rule:

		1	2	3	4	5	6	7	8
C1	Is today a weekday?	y	-	-	-				
C2	Is today a holiday?	n	y	y	y				
C3	Is it raining?	-	y	n	-				
A1	Go to work	1							
A2	Go on a picnic			1					
A3	Watch sports on TV		1						

LogicGem

The ambiguities were removed and 2 rules deleted.

OK

Condition Stub: 1

Completion Ratio: 8 : 10

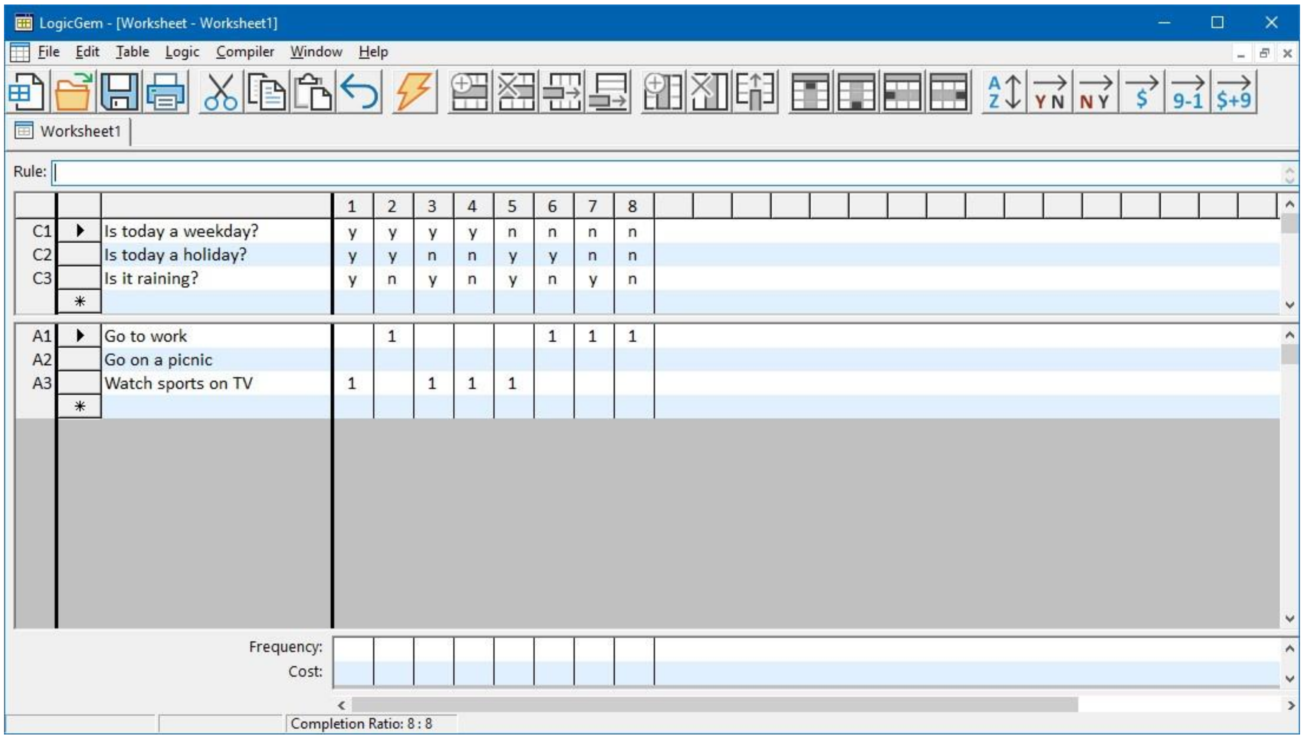
Figure 17 – Sample Ambiguous Decision Table

Expand

Expand is a logic engineering tool used to create the simple rules to represent all possible true/false combinations of conditions in the table. If action entries exist, then they will be expanded as well. The Expand function is useful when starting to fill out condition entries as it provides a big picture of the extent of the logic. Of course, a decision table with many conditions may yield a very large expanded table. One approach towards rule determination is to start with a fully expanded table, and then incrementally identify candidate rules for consolidation into few complex rules. After a series of iterations, a table of 64 expanded rules, for example, could be narrowed down to less than 10 rules.

NOTE: Invoking the Expand function may delete and replace previously entered rules.

Figure 18 shows an expanded version of our sample logic table which has only 4 complex rules in optimized form. The expanded table has 8 rules or 2^3 based on three conditions.


LogicGem - [Worksheet - Worksheet1]
File Edit Table Logic Compiler Window Help
Worksheet1
Rule:

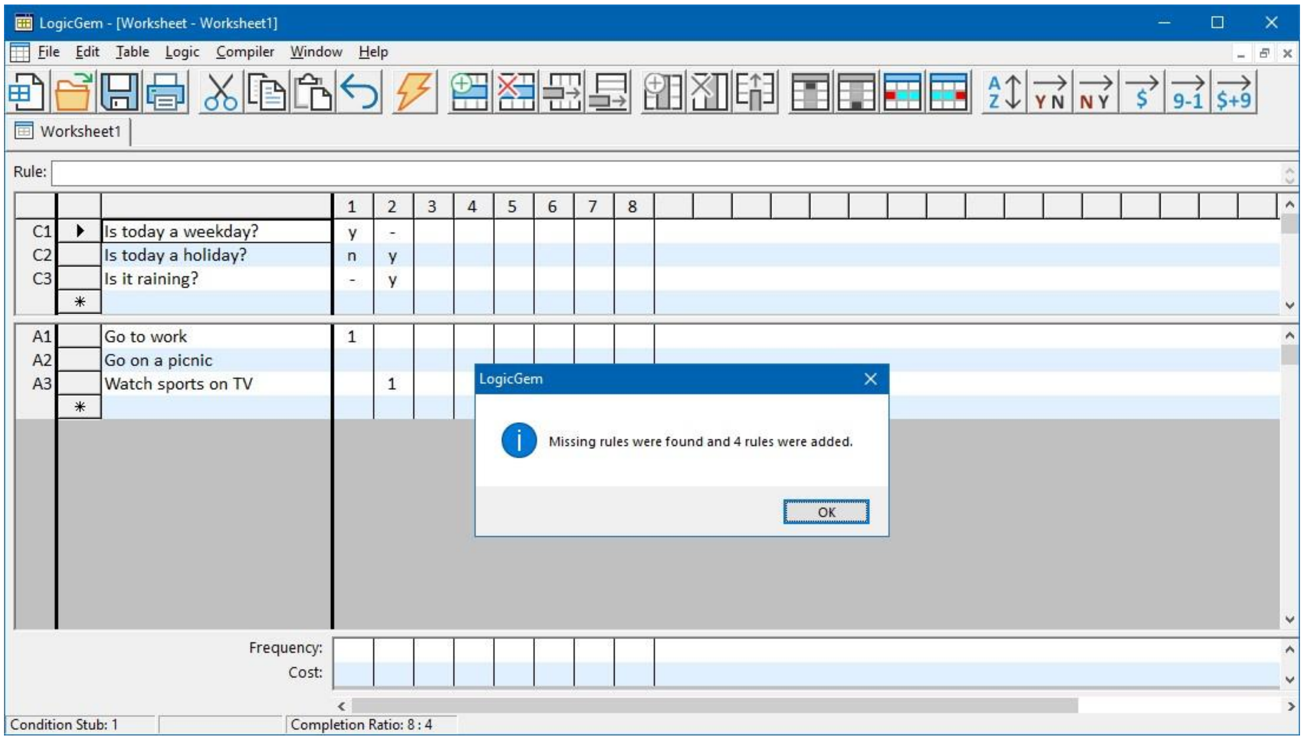
		1	2	3	4	5	6	7	8
C1	Is today a weekday?	y	y	y	y	n	n	n	n
C2	Is today a holiday?	y	y	n	n	y	y	n	n
C3	Is it raining?	y	n	y	n	y	n	y	n
A1	Go to work		1				1	1	1
A2	Go on a picnic								
A3	Watch sports on TV	1		1	1	1			

Frequency:
Cost:
Completion Ratio: 8 : 8

Figure 18 – Expanded Decision Table

Missing

Missing is a logic engineering tool used to indicate whether any rules are missing in a decision table and then adds the rules to complete the table. The added rules, however, are only in simple form (i.e. no complex rule optimizations are done). This function does not affect the rules that already exist in the table when it is invoked. The Missing rule is of tremendous value in helping to fill out the structure of large logic tables. One approach to logic table design is to start with a few well known rules, and then use the Missing function to help identify all remaining rules. One of the main strengths of LogicGem is that it contains tools like Missing to help you become aware of less well known rules that are required to make the logic table complete. If you use the Missing function on a logically complete table, no action is taken.

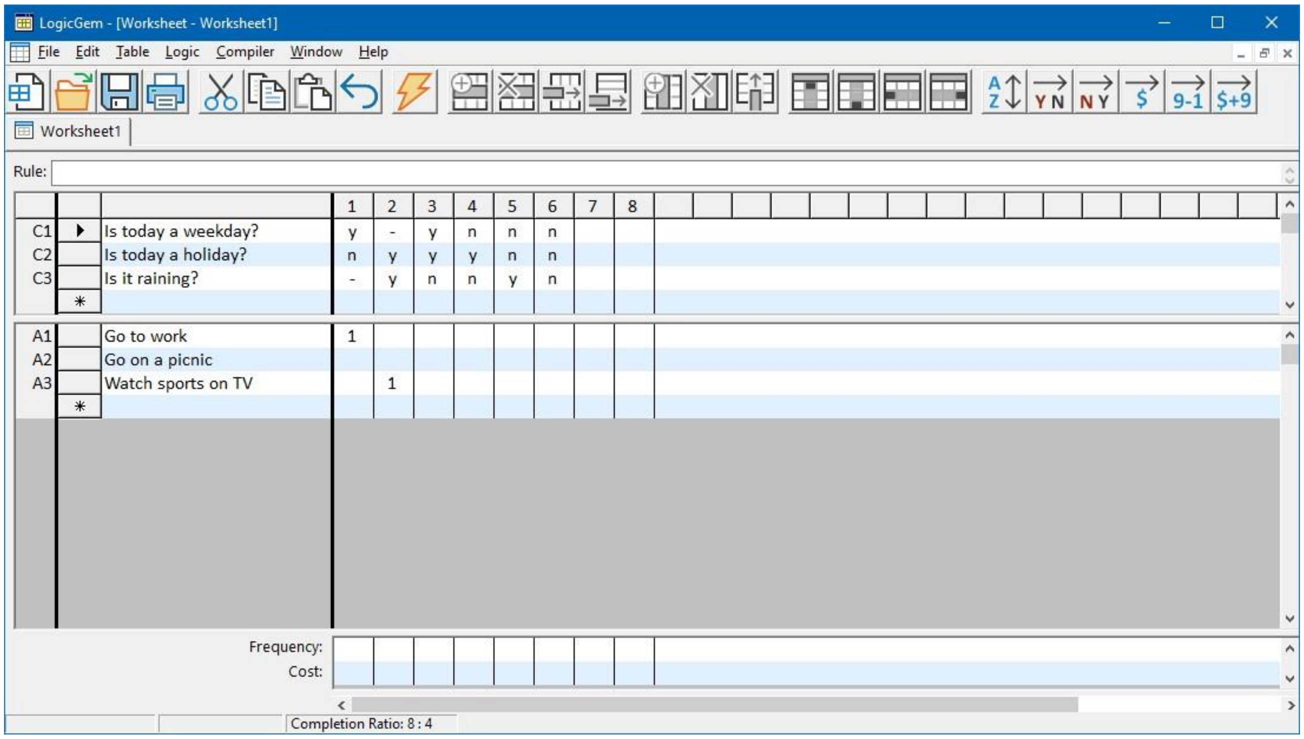


		1	2	3	4	5	6	7	8
C1	Is today a weekday?	y	-						
C2	Is today a holiday?	n	y						
C3	Is it raining?	-	y						
A1	Go to work	1							
A2	Go on a picnic								
A3	Watch sports on TV		1						

Missing rules were found and 4 rules were added.

Figure 19 – Using the Missing Function on Incomplete Logic Table

Consider the example table shown in Figure 19. The logic engineer is building a new table, but can only identify two well known complex rules. The engineer might have developed a case of “logic block” and can’t easily think of any other rules. This is where the Missing function comes in handy, and as shown, 4 additional rules are discovered. Figure 20 shows the same decision table, but this time with all missing rules included. From here, the task is to determine the action entries to complete the rules. Along the way of course, the missing rules may be consolidated into fewer complex rules.


LogicGem - [Worksheet - Worksheet1]
File Edit Table Logic Compiler Window Help
Worksheet1
Rule:
Frequency:
Cost:
Completion Ratio: 8 : 4

		1	2	3	4	5	6	7	8
C1	Is today a weekday?	y	-	y	n	n	n		
C2	Is today a holiday?	n	y	y	y	n	n		
C3	Is it raining?	-	y	n	n	y	n		
A1	Go to work	1							
A2	Go on a picnic								
A3	Watch sports on TV		1						

Figure 20 – Four Missing Rules Found

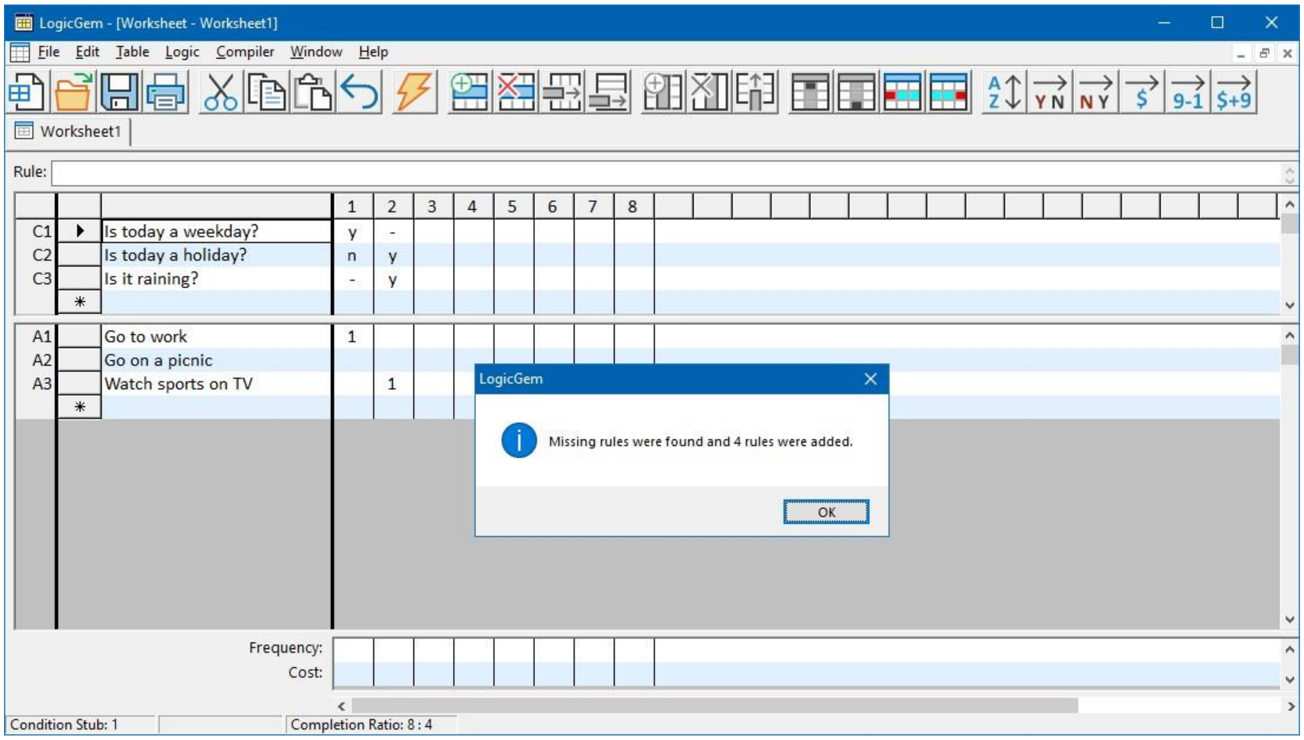
The Logic menu also includes a subset of the Missing function called "…One Rule at a Time" which identifies missing rules, but adds them to the decision table one at a time. You can use this function multiple times in succession to reveal all the missing rules. In larger tables, it may be less overwhelming to present just one missing rule at a time.

Reduce

Reduce is a logic engineering tool used to reduce two or more simple or complex rules into one complex rule. All the previous logic functions ignore the action rule entries in a table, which are execution-order priority numbers that connect an action to a rule. This is not the case with the logic Reduce function which acts to unite two rules into one single rule of greater complexity. The unification proceeds when the following criteria are met:

- The two candidate rules have the same action entry (entries)
- The two candidate rules have identical condition-row entries except for the entries on one of the condition rows.

If a pair of such rules exists, the Reduce function merges them into one single rule which is identical to them except on the condition-row where they differ. On this row the entry of the product rule is a dash, "-". In Figure 21, our sample logic table shows two rules that are candidates for reduction, Rules 3 and 4. Both criteria for reduction are satisfied: both rules have the same action entry: A2, and the rules differ only by the condition entries in C1. After using the Reduce function we get our original sample table with a complex Rule 3. After using Reduce, the Completion Ratio should remain unchanged. You may find that the Reduce tool does not reduce tables that are ambiguous. Please make sure your table is not ambiguous before using Reduce.


LogicGem - [Worksheet - Worksheet1]

		1	2	3	4	5	6	7	8
C1	Is today a weekday?	y	-						
C2	Is today a holiday?	n	y						
C3	Is it raining?	-	y						
A1	Go to work	1							
A2	Go on a picnic								
A3	Watch sports on TV		1						

LogicGem

Missing rules were found and 4 rules were added.

OK

Condition Stub: 1 | Completion Ratio: 8 : 4

Figure 21 – Reducing Two Simple Rules Into One Complex Rule

Compiling Decision Tables

Introduction to Compiling

The LogicGem Compiler menu provides tools that complete the decision table process by encoding the finished table in one of the supported programming or natural languages. The programming code generated by LogicGem can be copied/pasted into a programming editor and compiled into executable code for the target computing environment. LogicGem is entirely operating system and language compiler agnostic since the language constructs generated by the LogicGem Compiler tools are very generic in their syntax.

The Compiler menu includes sub-menus that set attributes for the compilation process. These attributes must be reviewed and set before you compile a decision table. They can be set one at a time from the Compiler menu or together from the Preferences dialog box (Refer to the Compiler Defaults section in the Preferences chapter for more information). After an initial compile is done, its attributes can be reset from the Compile window that displays the generated source code.

All compiler properties persist until you change them again even if you close LogicGem. The Compiler Tools table below itemizes all the sub-menus and selections available to the logic engineer to compiling decision tables.

In the compiler examples that follow, we'll use our sample decision table that is now *mechanically perfect* since there are no missing, contradictory or redundant rules. The table is finished and ready to compile.

Languages

The logic engineer can specify the Compiler generated code to be in one of several supported programming and natural languages (see Figure 22). A checkmark in the Language sub-menu indicates which programming or natural language to generate. Only one language may be specified at a time.

You may select the desired language using the following method:

Menu bar

Click on the Compiler menu function.

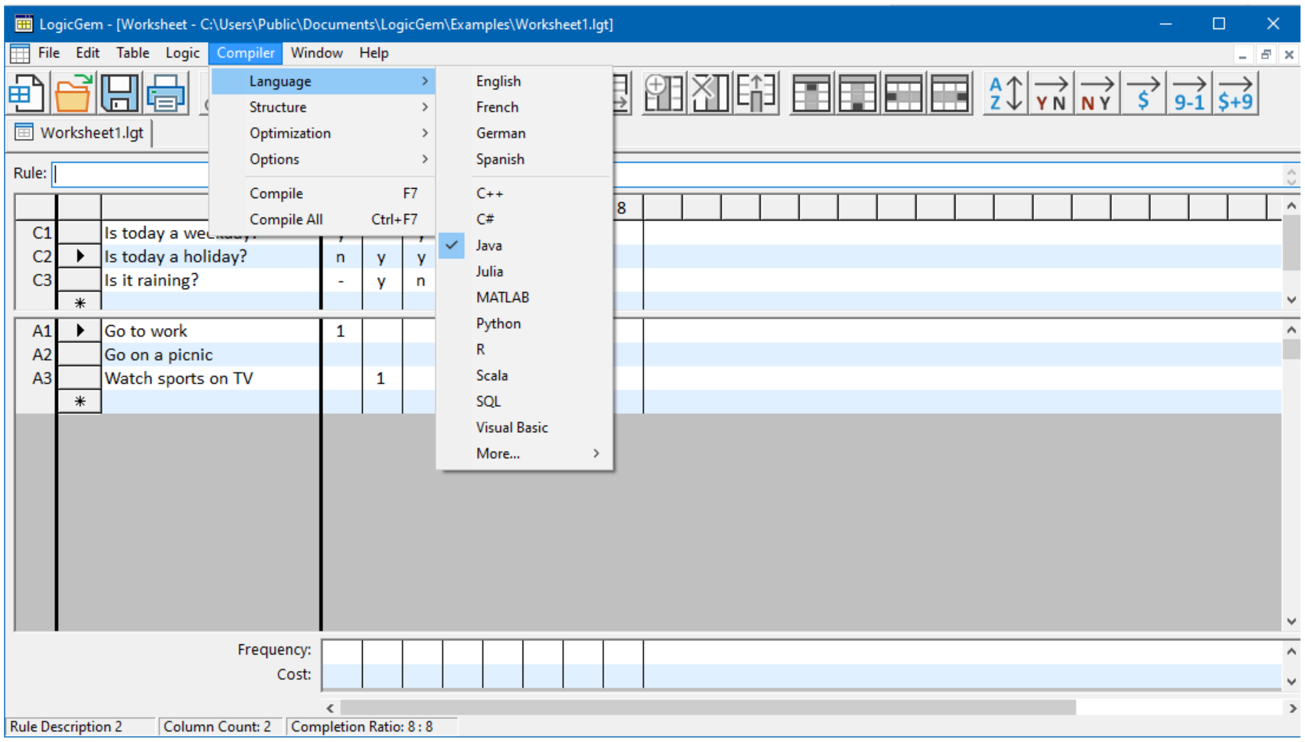
Click on the Language sub-menu to display the list of available languages.

Click on the desired language. The Compiler menu disappears after a selection is made. The next time you select the Language sub-menu, the language you selected will have a check mark.

Table 7 below lists all the languages supported by the LogicGem compiler along with the filename extensions used if you decide to save the compiled output.

Sub-Menu	File Extension
Natural Languages	
English	.txt
French	.txt
German	.txt
Spanish	.txt
Programming Languages	
BASIC	.bas
C	.c
C++	.cpp
COBOL	.cbl
Fortran	.for
FoxPro	.prg
Java	.java
JavaScript	.js
Julia	.jl
MatLab / Octave	.m
Pascal	.pas
Perl	.pl
PowerBuilder	.app
Python	.py
R	.r
SAS	.sas
Scala	.scala
SQL	.sql
Visual Basic	.vb
Visual Basic 6.0 (Classic)	.bas
xBase	.prg

Table 7 – Compiler Languages and File Extensions



C1	Is today a weekday?			
C2	Is today a holiday?	n	y	y
C3	Is it raining?	-	y	n
A1	Go to work	1		
A2	Go on a picnic			
A3	Watch sports on TV		1	

Figure 22 – The Compiler Menu with Language Sub-Menu

Structure

The logic engineer can specify the Compiler generated code to be in one of several structures: Nested If/Else, Case/Switch, and Rule List (see Figure 23). A checkmark in the Structure sub-menu indicates which structure to generate. Only one structure may be specified at a time. The specific syntax of the various structures supported by LogicGem depends on the target programming language, e.g. the Nested If/Else syntax for Visual Basic differs from Java.

You may select the desired structure using the following method:

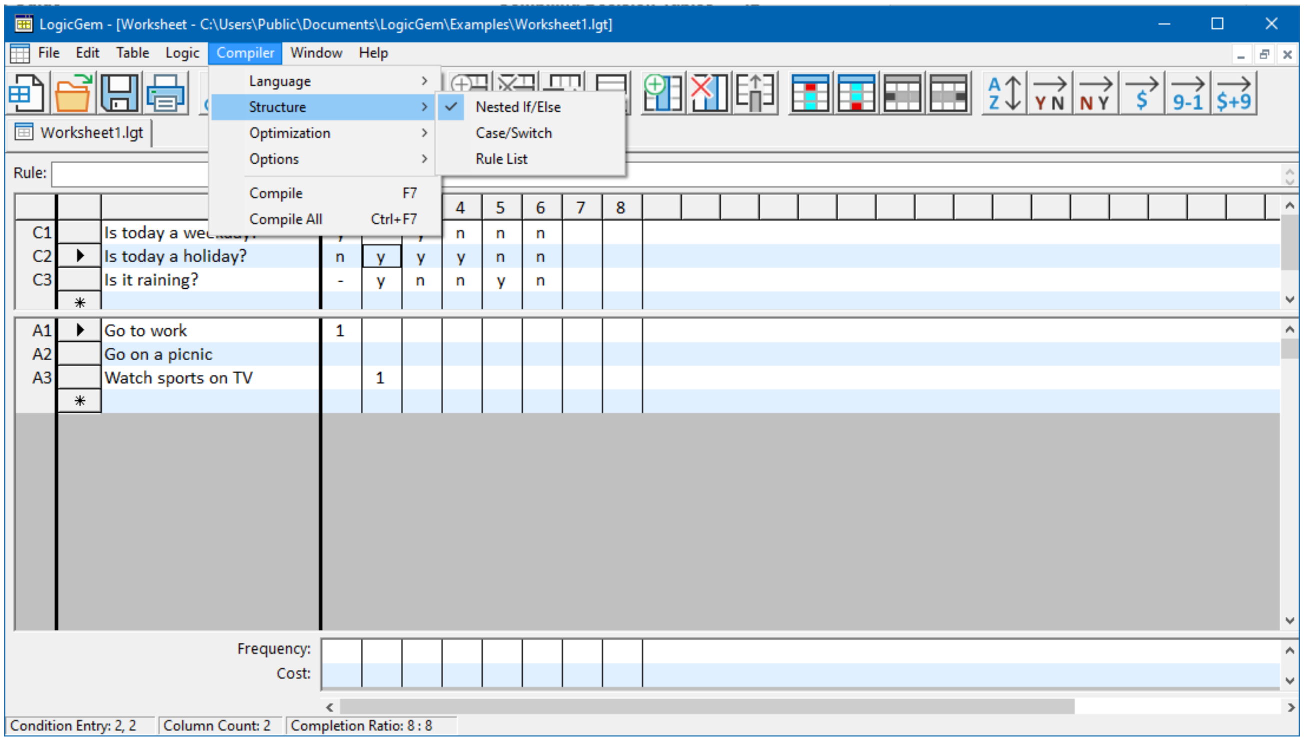
Menu bar

Click on the Compiler menu function.

Click on the Structure sub-menu to display the list of available structures.

Click on the desired structure. The Compiler menu disappears after a selection is made. The next time you select the Structure sub-menu, the structure you selected will have a check mark.

The “Nested If/Else” structure corresponds to the common nested if-then-else format of nested if-networks. Rules are not nested however in the “Rule List” format. All the necessary conditional values leading to a single rule are collectively grouped together and this rule-branch is presented separately from all the other rules. This is extremely useful to instantly verify all the required conditional inputs associated with a particular rule. The “Case/Switch” structure corresponds to the common case-switch format used in many contemporary programming languages such as C. Under this option each isolated rule is assigned a control variable (determined by LogicGem), which then directs program execution to an appropriate switch statement corresponding to the action(s) specified in the source logic table. The Case/Switch structure is often used for performance concerns. The “Rule List” format is commonly used with natural language compilation for business rule documentation purposes, since rules are easier to read in a list.



						4	5	6	7	8
C1		Is today a we	y	[illegible]	y	n	n	n		
C2	▶	Is today a holiday?	n	y	y	y	n	n		
C3		Is it raining?	-	y	n	n	y	n		
	*									
A1	▶	Go to work	1							
A2		Go on a picnic								
A3		Watch sports on TV		1						
	*									

Condition Entry: 2, 2 | Column Count: 2 | Completion Ratio: 8 : 8

Figure 23 – The Compiler Menu with Structure Sub-Menu

Options

The logic engineer can control the verbosity of the LogicGem Compiler by using the settings in the Options sub-menu of the Compiler menu. A checkmark in the Options sub-menu indicates which options are in effect during the compilation process. Any number of options may be in effect at a time.

This function allows the logic engineer to select the level and type of verbiage to be included in the Compiler output. Figure 24 shows the opened Options menu with the options: Rule Numbers, Statistic Code, Table, and Rule Descriptions.

Rule Numbers - With this option the rule numbers are included as comments in the Compiler output code. Cross reference can then be made with the source table or the comment-form table version outputted with the generated code to alleviate the task of trouble-shooting and understanding the code logic.

Statistic Code – This interesting and useful feature endows the Compiler generated source code with the ability to record the actual rule frequencies (when the code is executed and a particular rule is encountered its frequency is stepped up by one) and costs (a rule’s cost is stepped up by a user-determined functional value). This is accomplished by readily identifiable internal registers introduced in code. The user-programmer is required to interface with these registers to collect the data. If nothing else, these data serve as useful feedback to periodically refresh the source code either through modifying the logic table Frequency and Cost inputs or through updating the user-provided cost function.

For example, generated compiled code output might include the following statements for Rule 9 of a given decision table (since index values start at zero, the index value of 8 indicates Rule 9):

```
Frequency(8) = Frequency(8) + 1
Cost(8) = Cost(8) + C(8)
```

Here, you will have to declare the register arrays Frequency() and Cost() since the LogicGem compiler doesn’t provide such code. You’ll also have to write your own cost function C(). In addition, you must write your own code to utilize the final frequency and cost values after the code has executed. For example, you may wish to record the values in a log file or database.

Table – A comment-form version of the actual source logic table is appended to the Compiler output code. This permanent attachment of the table to the code is very useful documenting tool since the actual source table can be readily recreated from its comment-form version.

Rule Names – With this option the user-defined rule-names are included as comments in the Compiler output code. As with the case of the "Rule Number" option this allows quick cross-reference with other documented source, the English documentation for example, for easier trouble-shooting and logic-tracing the code.

You may select the desired options using the following method:

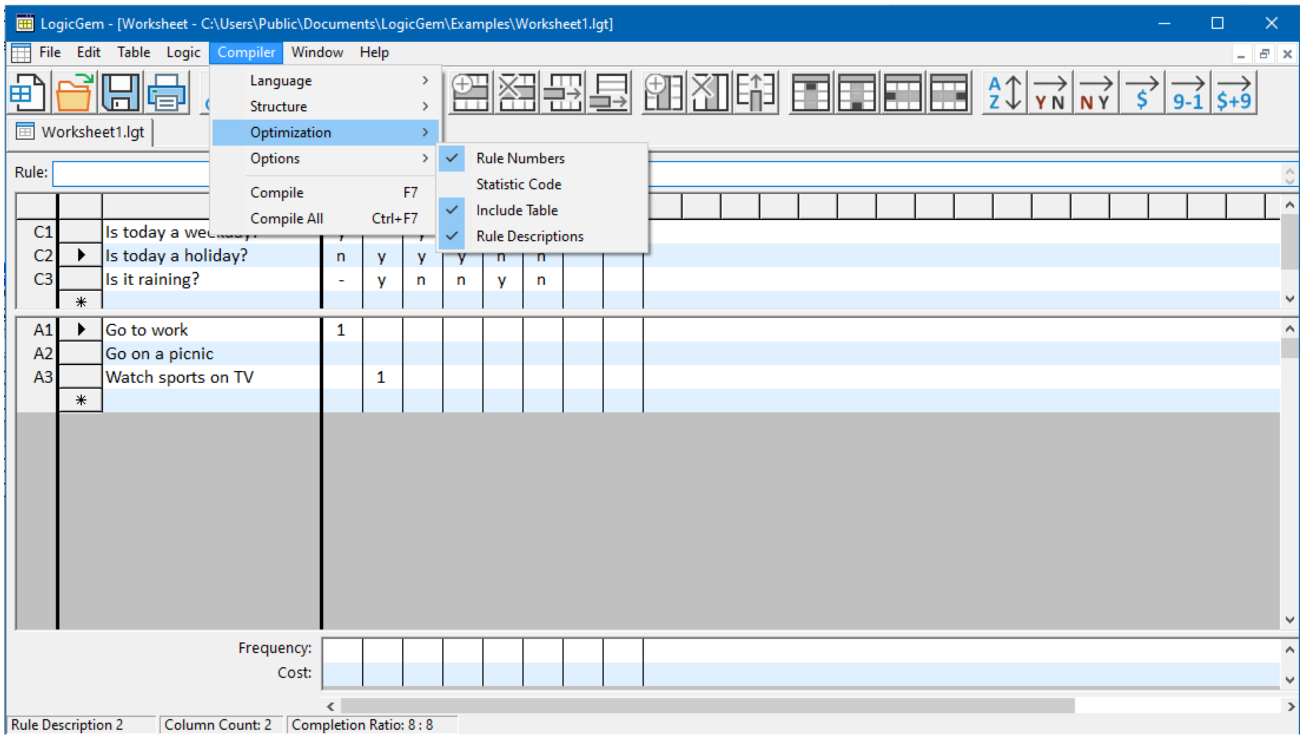
Menu bar

Click on the Compiler menu function.

Click on the Options sub-menu to display the list of available options.

Click on the desired option. The Compiler menu disappears after a selection is made. The next time you select the Options sub-menu, the option you selected will have a check mark. The checkmark(s) indicates which compiler option(s) to include with the compiled output.

Repeat the above process for each additional option.



✓ Rule Numbers
Statistic Code
✓ Include Table
✓ Rule Descriptions

C1	Is today a wee	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	
C2	Is today a holiday?	n	y	y	y	n	n	
C3	Is it raining?	-	y	n	n	y	n	
A1	Go to work	1						
A2	Go on a picnic							
A3	Watch sports on TV		1					

Figure 24 – The Compiler Menu with Options Sub-Menu

Optimization

For optimization and analysis purposes, it is often useful to sort the compiled output of a logic table in a number of different ways. The logic engineer can control how the code is generated by the LogicGem Compiler by using the options in the Optimization sub-menu of the Compiler menu. A checkmark in the Optimization sub-menu indicates which sort option is in effect during the compilation process. Only one option is in effect at a time.

NOTE: It is recommended that most projects use the default optimization. Due to its inherent nature, the optimization options are generally only appropriate for the Nested If/Else code structure.

Figure 25 shows the opened Optimization sub-menu with the options:

- **Default** – The rules are prioritized as they are listed in the table.
- **Prioritize Yes Columns** – Yes columns are prioritized before the No columns.
- **Prioritize No Columns** – No columns are prioritized before the Yes columns.
- **Prioritize Code** – Rules are prioritized according to their increasing value assigned in the Cost option.
- **Prioritize Frequency** – Rules are prioritized according to their decreasing value assigned in the Frequency option.
- **Cost / Frequency Heuristics** – Rules are prioritized according to their decreasing value of combined cost-frequency ratio.

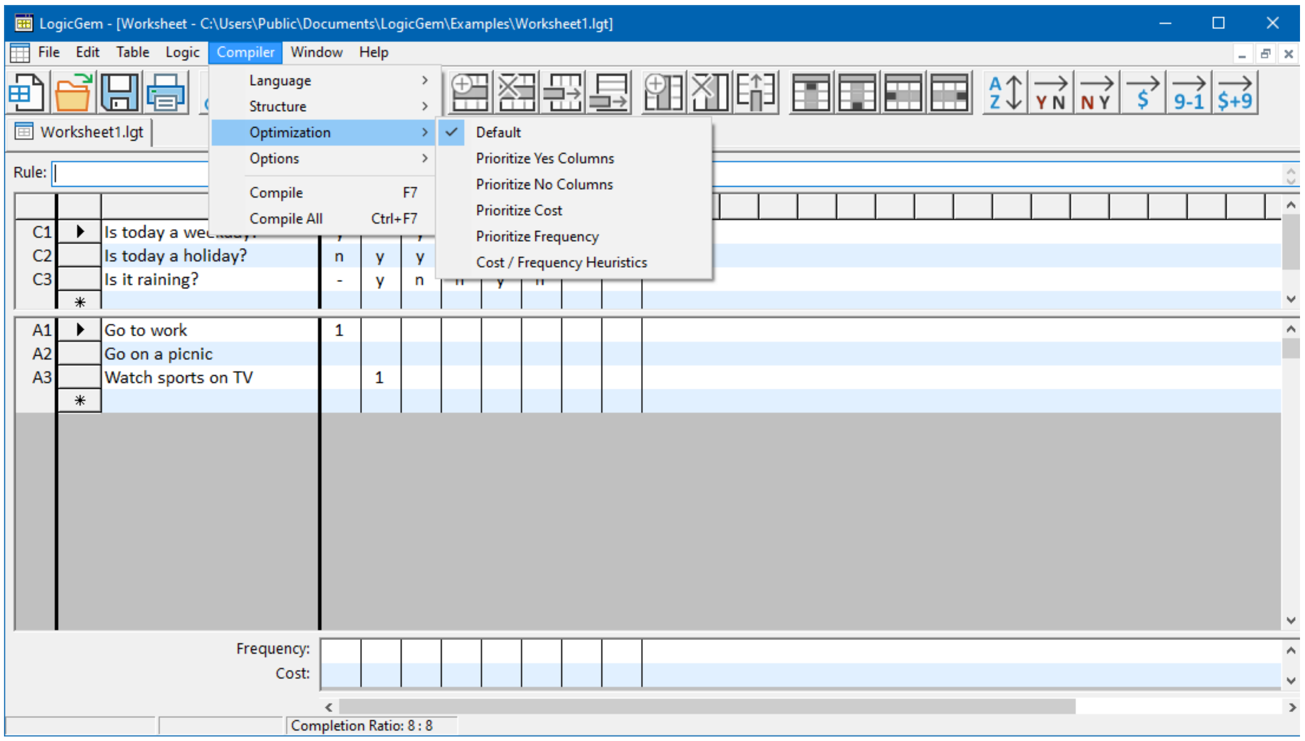
You may select the desired Optimization option using the following method:

Menu bar

Click on the Compiler menu function.

Click on the Optimization sub-menu to display the list of available options.

Click on the desired option. The Compiler menu disappears after a selection is made. The next time you select the Optimization sub-menu, the option you selected will have a check mark.



C1	Is today a we[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	
C2	Is today a holiday?	n	y	y	[illegible]	[illegible]	[illegible]	
C3	Is it raining?	-	y	n	n	y	n	
A1	Go to work	1						
A2	Go on a picnic							
A3	Watch sports on TV		1					

Figure 25 – The Compiler Menu with Optimization Sub-Menu

Compile

Selecting the Compile option on the Compiler menu invokes the LogicGem language compiler which translates the logic table into a desired natural or programming language. Source code is generated immediately for the active Worksheet and displayed in a Compile window. The generated code file is automatically saved into your table directory with an appropriate file extension. The file name is displayed in the Compile window’s title bar.

You may invoke the LogicGem compiler using one of the following methods:

Menu bar

Click on the Compiler menu function.

Click on the Compile option.

View compiled code in the Compile window.

Tool Bar

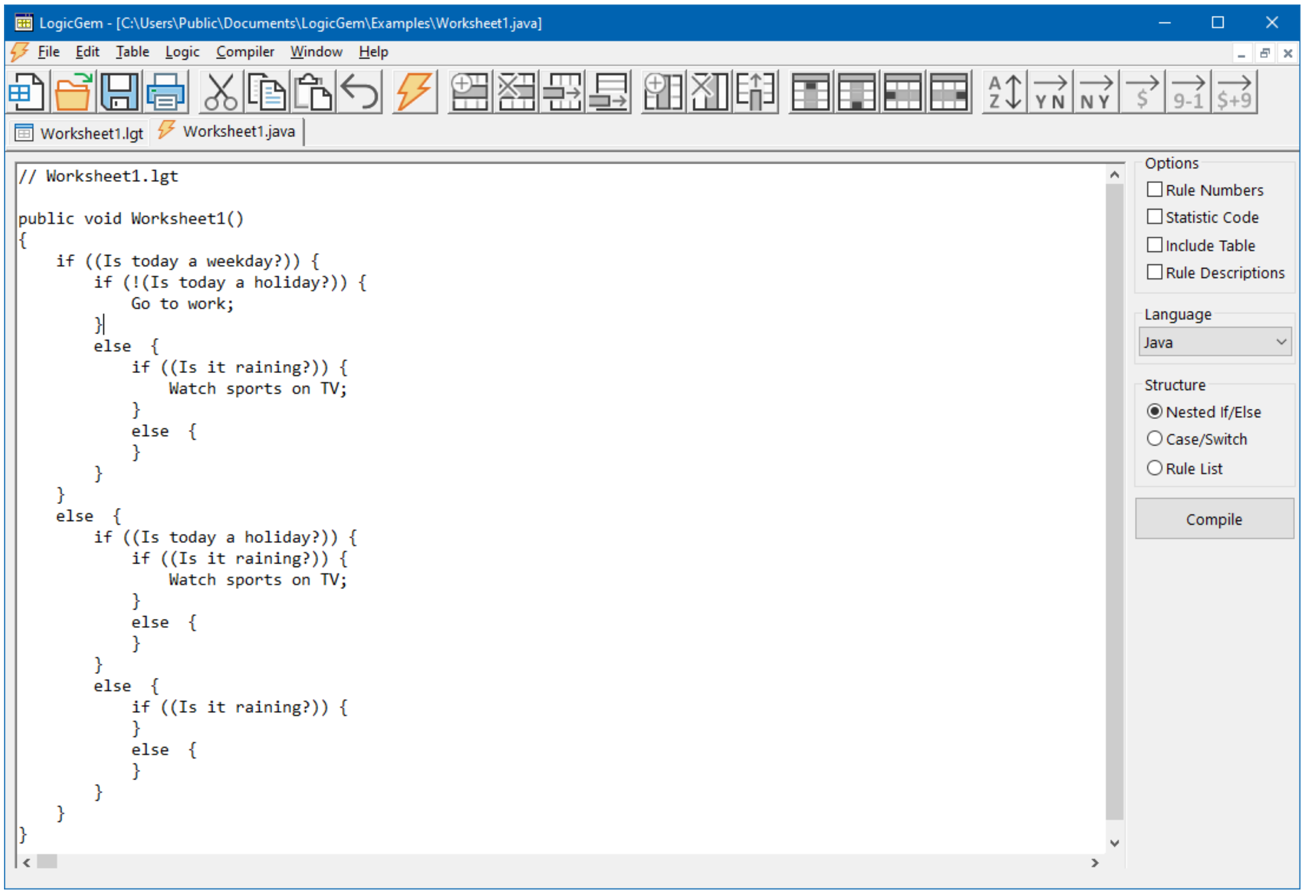
Click on the icon to start a compile.

The Compiler menu also offers the Compile All option that generates source code immediately for the active and all inactive Worksheets. The generated code files are automatically saved into your table directory with the appropriate file extensions. Multiple Compile windows are opened, one for each Worksheet.

Figure 26 shows a compile of the sample decision table using the following parameters:

- Language: Java
- Structure: Nested If/Else
- Options: all compiler verbosity options turned off
- Sort: No sort

Since the condition and action stubs in the sample decision table don’t have actual Java programming code in them, the code generated by LogicGem in this case should only be considered pseudo-code. This example illustrates the need for the logic programmer to be diligent in using valid programming language constructs in decision table stubs if LogicGem is to be used for generating programming code. Using LogicGem to develop generic business rules, on the other hand, allows the flexibility of using simple phrases for stub values.


LogicGem - [C:\Users\Public\Documents\LogicGem\Examples\Worksheet1.java]
File Edit Table Logic Compiler Window Help
Worksheet1.lgt
Worksheet1.java
Options
Language
Structure
Compile

```
// Worksheet1.lgt

public void Worksheet1()
{
    if ((Is today a weekday?)) {
        if (!(Is today a holiday?)) {
            Go to work;
        }
        else  {
            if ((Is it raining?)) {
                Watch sports on TV;
            }
            else  {
            }
        }
    }
    else  {
        if ((Is today a holiday?)) {
            if ((Is it raining?)) {
                Watch sports on TV;
            }
            else  {
            }
        }
        else  {
            if ((Is it raining?)) {
            }
            else  {
            }
        }
    }
}
```

Options: ☐ Rule Numbers, ☐ Statistic Code, ☐ Include Table, ☐ Rule Descriptions

Language: Java

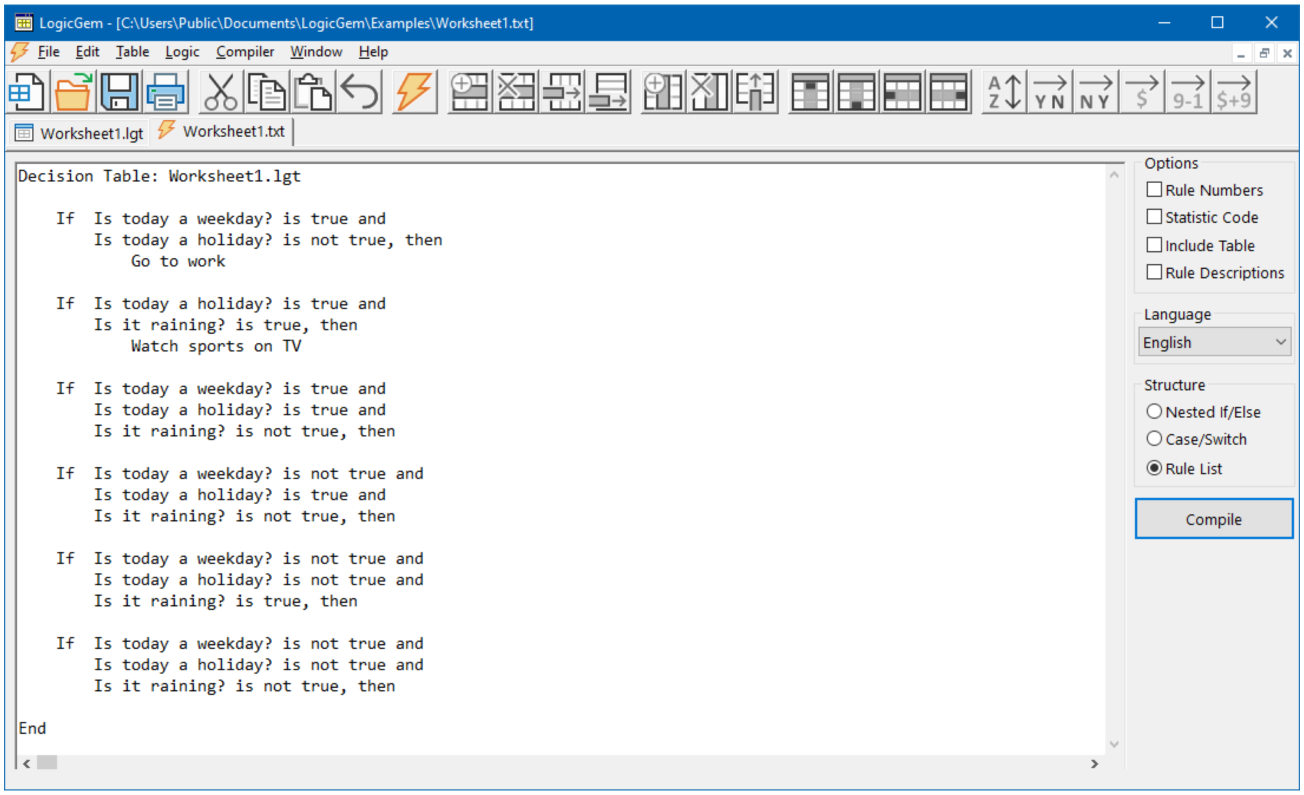
Structure: [✓ Nested If/Else] Case/Switch Rule List

Figure 26 – Sample Decision Table Compiled in Java

Figure 27 shows a compile of the sample decision table using the following parameters:

- Language: English
- Structure: Rule List
- Options: all compiler verbosity options turned off
- Sort: No sort

This type of compile is ideal for system and/or programming documentation. Since documenting the core logic of a software application is often difficult and time consuming, this LogicGem feature can result in significant time savings.


LogicGem - [C:\Users\Public\Documents\LogicGem\Examples\Worksheet1.txt]
File Edit Table Logic Compiler Window Help
Worksheet1.lgt
Worksheet1.txt
Options
Language
Structure
Compile

```
Decision Table: Worksheet1.lgt

    If  Is today a weekday? is true and
        Is today a holiday? is not true, then
            Go to work

    If  Is today a holiday? is true and
        Is it raining? is true, then
            Watch sports on TV

    If  Is today a weekday? is true and
        Is today a holiday? is true and
        Is it raining? is not true, then

    If  Is today a weekday? is not true and
        Is today a holiday? is true and
        Is it raining? is not true, then

    If  Is today a weekday? is not true and
        Is today a holiday? is not true and
        Is it raining? is true, then

    If  Is today a weekday? is not true and
        Is today a holiday? is not true and
        Is it raining? is not true, then

End
```

Options: [ ] Rule Numbers, [ ] Statistic Code, [ ] Include Table, [ ] Rule Descriptions

Language: English

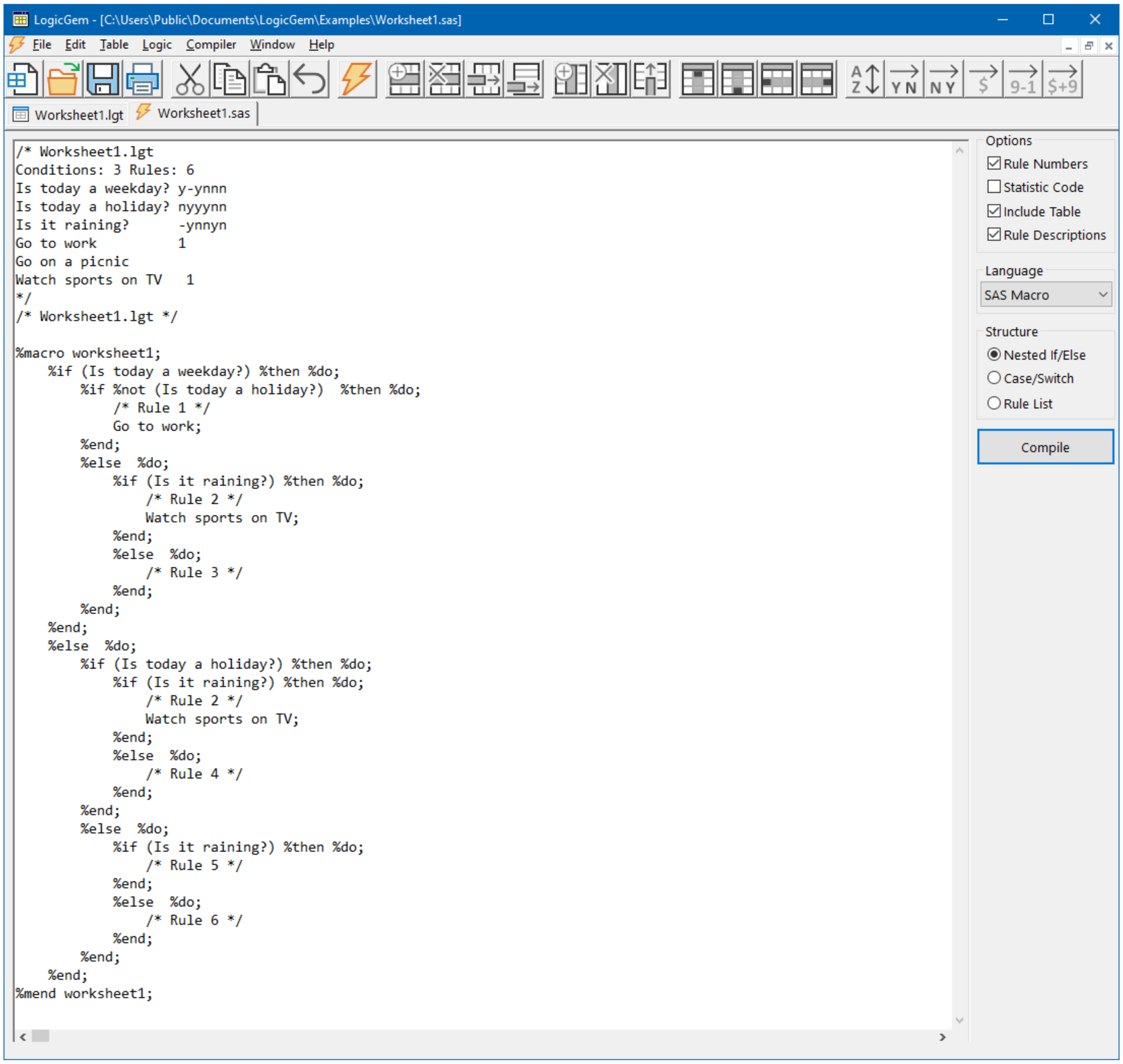
Structure: ( ) Nested If/Else, ( ) Case/Switch, [✓ Rule List]

Figure 27 – Sample Decision Table Compiled in English

Figure 28 shows a compile of the sample decision table using the following parameters:

- Language: SAS
- Structure: Nested If/Else
- Options: Rule Numbers, Table, and Rule Descriptions
- Sort: No sort

For this compilation, we've used a higher degree of verbosity, specifically Rule Numbers, Table, and Rule Descriptions. The result is more readable and better documented pseudo-code.


LogicGem - [C:\Users\Public\Documents\LogicGem\Examples\Worksheet1.sas]
File Edit Table Logic Compiler Window Help
Worksheet1.lgt
Worksheet1.sas
Compile

```
/* Worksheet1.lgt
Conditions: 3 Rules: 6
Is today a weekday? y-ynnn
Is today a holiday? nyyynn
Is it raining?      -ynnyn
Go to work          1
Go on a picnic
Watch sports on TV   1
*/
/* Worksheet1.lgt */

%macro worksheet1;
    %if (Is today a weekday?) %then %do;
        %if %not (Is today a holiday?)  %then %do;
            /* Rule 1 */
            Go to work;
        %end;
        %else  %do;
            %if (Is it raining?) %then %do;
                /* Rule 2 */
                Watch sports on TV;
            %end;
            %else  %do;
                /* Rule 3 */
            %end;
        %end;
    %end;
    %else  %do;
        %if (Is today a holiday?) %then %do;
            %if (Is it raining?) %then %do;
                /* Rule 2 */
                Watch sports on TV;
            %end;
            %else  %do;
                /* Rule 4 */
            %end;
        %end;
        %else  %do;
            %if (Is it raining?) %then %do;
                /* Rule 5 */
            %end;
            %else  %do;
                /* Rule 6 */
            %end;
        %end;
    %end;
%mend worksheet1;
```

Options: [✓ Rule Numbers] Statistic Code [✓ Include Table] [✓ Rule Descriptions]

Language: SAS Macro

Structure: [✓ Nested If/Else] Case/Switch Rule List

Figure 28 – Sample Decision Table Compiled in SAS

Code Generation and Optimization

The total number of possible programs which can be generated from a limited entry table of (n) conditions is given by the formula:

$$\text{possible programs} = \prod_{i=1}^{n} i^{2^{(n-i)}}$$

This means that for a simple set of five (y/n) conditions, there are 1,658,880 possible programs. The thought of inspecting all of them by hand to select the optimal generated code should tell you why you need LogicGem.

WARNING: LogicGem will attempt to generate code from incomplete or imperfect logic tables. Code generated from such tables will certainly not be correct, much less optimal. Many times it will not have correct syntax and will not compile using a commercial language compiler product.

LogicGem cannot understand the semantic content of your code and therefore might generate code which you can optimize further by hand. In particular, you can take a step towards optimization by looking at the lowest or deepest level of nesting in the if/then/else statements for duplicated actions. For example, consider the code fragment:

```
IF (Cx)
THEN Action
ELSE Action;
```

LogicGem has no way of knowing if the Action module will work the same way when condition Cx is TRUE as it will when Cx is FALSE. It has to take the safest course and generate the full decision structure. As the logic programmer, you should know if Action really needs condition Cx to operate as intended. If it does not care, then this if/then/else statement can be optimized to a single call to the Action module. This sort of code structure usually happens when the user has added a special “error” action to complete the missing actions in a logic table.

Another possible optimization in the generated code is where nested if/then/else statements can be replaced by Boolean operators that are supported by the target programming language. For example, consider the code fragment:

```
IF (Cx)
THEN IF (Cy)
      THEN Action1
      ELSE Action2
ELSE Action2;
```

LogicGem has no way of knowing if this can be optimized to the following shorter and more concise version with a Boolean AND operator:

```
IF (Cx AND Cy)
THEN Action1
ELSE Action2;
```

In a similar manner, consider the fragment:

```
IF (Cx)
THEN Action1
ELSE IF (Cy)
      THEN Action1
      ELSE Action2;
```

This code can often, but not always, be optimized to:

```
IF (Cx OR Cy)
THEN Action1
ELSE Action2;
```

It is worth noting that a good optimizing compiler will convert Boolean expressions into the object code equivalent of the original LogicGem nested if/then/else structures. This is called short circuit evaluation because it arrives at an action with the smallest number of logical tests. A weaker compiler will not do these optimizations, so the original LogicGem generated code will actually run faster than the hand tuned versions.

If execution speed is important, you might want to compare the original against a hand tuned version, and then pick the fastest one. If execution speed is not important, hand tuned code might be easier for people to read because it will not have such deeply nested program control structures.

There are similar optimizations for case/switch statements. Unfortunately, these are more language dependent For example the Pascal CASE statement differs in structure from the C switch statement.

In the final analysis however, the original code resulting from a LogicGem compile of a perfected logic table will always work fine. Optimizations such as those described above are entirely optional.

Samples

Business Rule Examples

This section provides two examples of how to use LogicGem for creating bullet proof business logic. In each case, we'll start with a description of the business requirements in general terms (the kind you typically get in a strategy meeting) and then demonstrate how to incrementally build a decision table to represent the rules that represent the requirements. Once the table is logically complete, we'll finish up by generating the English language rule-set that can be the basis of software documentation.

Example 1: RFID Wine Distribution Center

The first business rules example will demonstrate how LogicGem can be used to solve a simple logistics problem for a wine distribution center equipped with radio frequency identifier (RFID) technology.

The scenario goes as follows. A wine distribution business receives cases of wine bottles in a warehouse. There are three lines of wine received at a receiving dock. Each line has a unique stock keeping unit (SKU). Within the warehouse, there is one assembly line for each SKU where workers provide special packaging and premiums for retail stores.

The business rules are simple; for each different SKU, activate a special forklift that transports the cases to the appropriate assembly line. In addition, there is a temperature sensor at the receiving dock that determines whether the temperature is outside the optimal range for wine storage. If so, the case needs to be checked for spoilage before being transported to the assembly line. In the case of SKU #303, we need to check for spoilage regardless of the sensor's reading (this particular wine has special needs).

Here are the conditions we need to consider in this simple example:

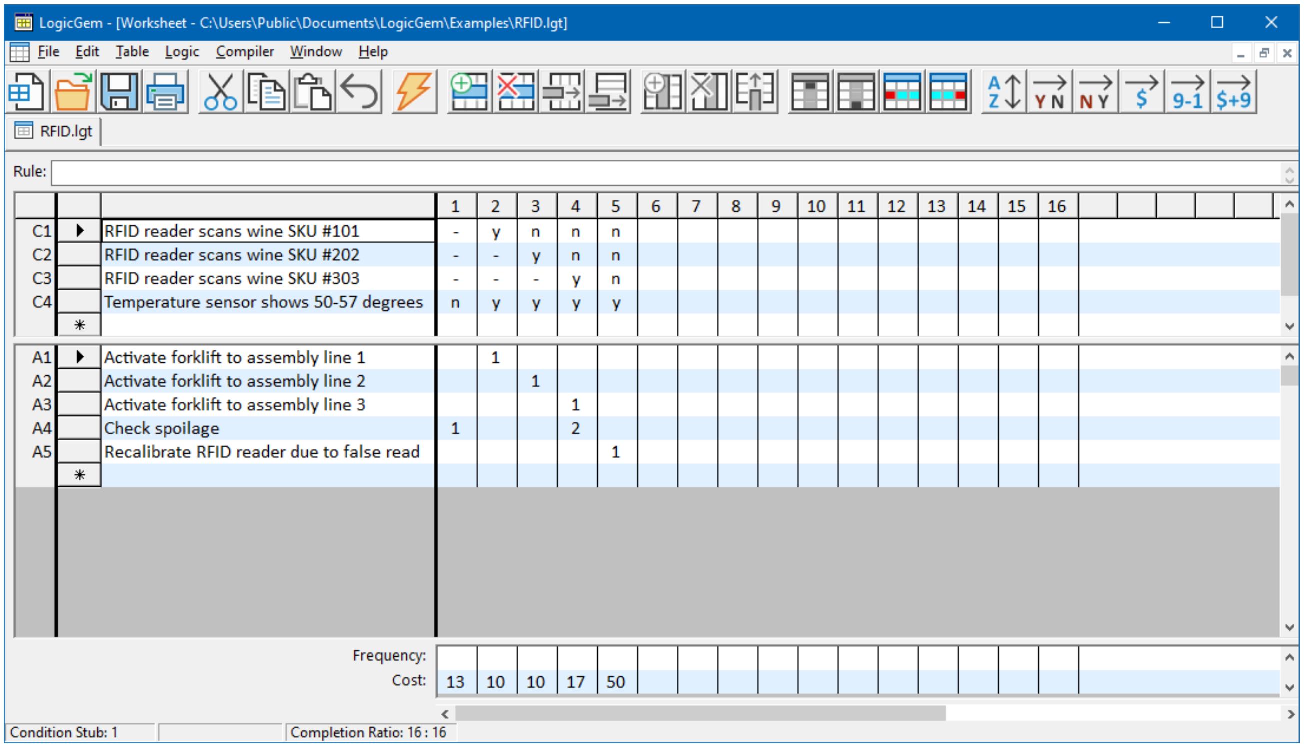
- RFID reader scans wine SKU #101
- RFID reader scans wine SKU #202
- RFID reader scans wine SKU #303
- Temperature sensor shows 50-57 degrees

Here are the actions we need to consider in this simple example:

- Activate forklift to assembly line 1
- Activate forklift to assembly line 2
- Activate forklift to assembly line 3
- Check spoilage
- Recalibrate RFID reader due to false read

The resulting decision table is shown in Figure 29. You can load a completed version of this table by selecting **File** | **Open**, highlighting the rfid.lgt table file in the LogicGem Examples folder, and clicking the Open button.

Let's consider the rules to see how the actions are tied to conditions. Rule 1 is the simplest as it considers the case where the temperature sensor is out of range. So regardless of the values of the other conditions ("don't care" values), if the value for C4 is N, then we carry out the action to check spoilage. Rule 4 is interesting because there are two actions to carry out. First, if conditions C3 and C4 are both Y, then we need to activate the forklift for assembly line 3, and check spoilage. Rule 5 is sort of a catch-all rule accommodating the case where the RFID reader reads a case, but does not recognize it as a valid SKU.


LogicGem - [Worksheet - C:\Users\Public\Documents\LogicGem\Examples\RFID.lgt]
File Edit Table Logic Compiler Window Help
RFID.lgt
Rule:

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
C1	RFID reader scans wine SKU #101	-	y	n	n	n											
C2	RFID reader scans wine SKU #202	-	-	y	n	n											
C3	RFID reader scans wine SKU #303	-	-	-	y	n											
C4	Temperature sensor shows 50-57 degrees	n	y	y	y	y											
A1	Activate forklift to assembly line 1		1														
A2	Activate forklift to assembly line 2			1													
A3	Activate forklift to assembly line 3				1												
A4	Check spoilage	1			2												
A5	Recalibrate RFID reader due to false read					1											
	Frequency:																
	Cost:	13	10	10	17	50											

Condition Stub: 1 Completion Ratio: 16 : 16

Figure 29

Cost: you're able to enter a two digit number in the cost row indicating a relative cost estimate assigned to a specific rule. Cost is a subjective measure of how much a rule will cost to execute.

Notice that we entered some integer values in the cost row at the bottom of the decision table. Assigning weighted costs to each rule adds another dimension to how LogicGem can optimize business logic. In this case, we assign a relatively high value of 50 to the rule where recalibration of the RFID reader is needed, and relatively low values of 10 to simple rules where only forklift activation is required. LogicGem's logic compiler is able to sort the resulting business logic by cost to obtain a potentially more optimal view of the logic.

Let's see what our business logic looks like by asking LogicGem to use the logic compiler to convert the decision table to English language rules. Start by selecting the **Compiler** | **Language** menu and select "English" and then selecting **Compiler** | **Structure** | **Rule List** (a good option for compiling decision tables representing business requirements as opposed to programming code). Lastly, select **Compiler** | **Options** menu and turn on Rule Numbers and Rule Descriptions (turn on the right hand side check boxes).

Next select **Compiler** | **Compile** or press the **F7** function key to tell LogicGem to compile the decision table in order to yield business rules. See the English business rules below.

```
Procedure: rfid.lgt
```

```
1
If  Temperature sensor shows 50-57 degrees is not true , then
             rule 1
             *** When sensor show out-of-range, must check for spoilage
            Check spoilage,

2
If  RFID reader scans wine SKU #101 is true and
    Temperature sensor shows 50-57 degrees is true , then
             rule 2
             *** Use assembly line 1 forklift to transport cases of SKU #101
            Activate forklift to assembly line 1,

3
If  RFID reader scans wine SKU #101 is not true and
    RFID reader scans wine SKU #202 is true and
    Temperature sensor shows 50-57 degrees is true , then
             rule 3
             *** Use assembly line 2 forklift to transport cases of SKU #202
            Activate forklift to assembly line 2,

4
If  RFID reader scans wine SKU #101 is not true and
    RFID reader scans wine SKU #202 is not true and
    RFID reader scans wine SKU #303 is true and
    Temperature sensor shows 50-57 degrees is true , then
             rule 4
             *** Use assembly line 3 forklift to transport cases of SKU #303,
                 and also check for spoilage
            Activate forklift to assembly line 3,
            Check spoilage,

 5
If  RFID reader scans wine SKU #101 is not true and
    RFID reader scans wine SKU #202 is not true and
    RFID reader scans wine SKU #303 is not true and
    Temperature sensor shows 50-57 degrees is true , then
             rule 5
             *** We got a false read from RFID reader, must recalibrate the
                 device
            Recalibrate RFID reader due to false read,
```

Example 2: Complex Shipping Charge Calculation

With the next business rules example, we'll describe a more complex set of business requirements to show how easily LogicGem can address intricate logic and provide the business analysts with a sense of confidence that the rule-set is complete.

The example we'll consider in this section is a shipping charge calculation. Determining a shipping charge to fulfill a product order is often comprised of a number of interrelated factors such as:

- Dollar value of the order
- Customer status (e.g. new vs. favored)
- Order weight
- Destination (e.g. domestic vs. international)
- Special handling requirements (e.g. hazardous materials)
- Carrier (e.g. Federal Express, Airborne Express, etc.)

For this example, we'll consider the following business rules, each playing a role in determining the shipping charge:

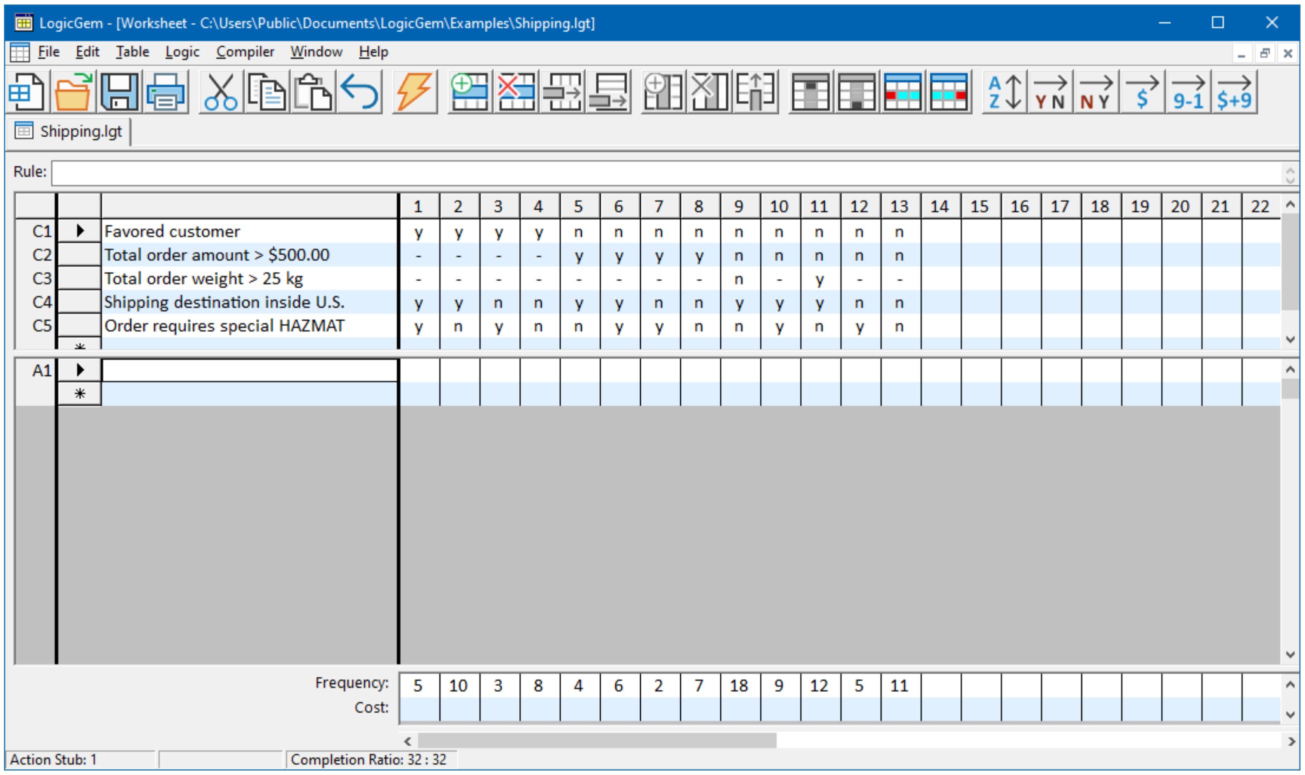
- Is it a favored customer?
- Is the total order amount more than $500?
- Is the total weight of the order more than 25kg?
- Domestic or international delivery?
- Does the order require special handling for hazardous materials?

You can load a completed version of this table by selecting **File** | **Open**, highlighting the shipping.lgt table file in the LogicGem Examples folder, and clicking the Open button.

Figure 30 shows the selected condition stubs and rules for this business problem. We can examine a few rules to make sure we understand the intent. First let's consider the first condition stub "Favored customer." If the current customer has favored status then shipping is easy, it's free. This means that the other conditions that determine the cost of shipping can be ignored using "don't cares" for C2 and C3 as shown in Rule 1. Rule 9 is interesting because we need to itemize all its Y or N values: not a favored customer, small order amount, light weight order, domestic destination, and no special handling. This rule is probably the most common rule due to all factors. Finally, Rule 7 is the most complex: not a favored customer, small order amount, international destination, and special handling required. In this case, the shipping business requirements say that weight isn't a factor in determining a charge.

You might also consider using the Rule field located just above the decision table area of the screen. Here you can write descriptive text for each individual rule. When you go to compile the decision table, you'll have the option of including the Rule text for the purpose of documenting the compiled table. Notice in Figure 30 that with Rule 7 selected, its rule description appears in this field.

Rule: You're able to enter descriptive text in the Rule field for each rule in the decision table.



LogicGem - [Worksheet - C:\Users\Public\Documents\LogicGem\Examples\Shipping.lgt]
File Edit Table Logic Compiler Window Help
Shipping.lgt

Rule:

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
C1	Favored customer	y	y	y	y	n	n	n	n	n	n	n	n	n									
C2	Total order amount > $500.00	-	-	-	-	y	y	y	y	n	n	n	n	n									
C3	Total order weight > 25 kg	-	-	-	-	-	-	-	-	n	-	y	-	-									
C4	Shipping destination inside U.S.	y	y	n	n	y	y	n	n	y	y	y	n	n									
C5	Order requires special HAZMAT	y	n	y	n	n	y	y	n	n	y	n	y	n									
A1																							
	Frequency:	5	10	3	8	4	6	2	7	18	9	12	5	11									
	Cost:																						

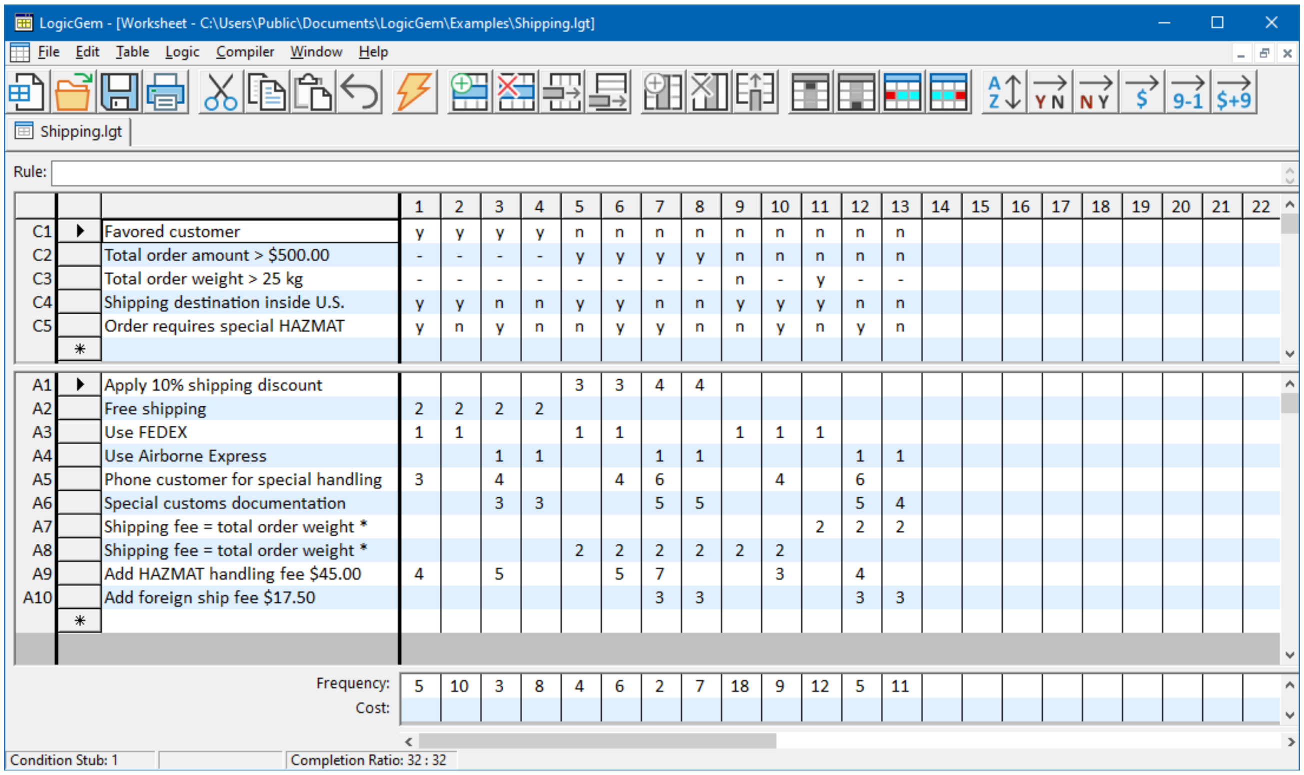
Action Stub: 1 | Completion Ratio: 32 : 32

Figure 30

Now let's review the completed decision table with the actions filled in as shown in Figure 31. The business requirements call for the following actions to be considered:

- Apply a 10% discount on the shipping charge.
- Free shipping.
- Use Federal Express as the shipping vendor.
- Use Airborne Express as the shipping vendor.
- Have customer service phone the customer to discuss special handling requirements.
- Special customs documentation for international destinations.
- Shipping fee calculation: total order weight times the factor $1.50/kg.
- Shipping fee calculation: total order weight times the factor $1.25/kg.
- Add $45 special handling fee for hazardous materials.
- Add $17.50 foreign shipping fee for international orders.

Going back to Rule 7, we see that seven actions are required to satisfy this rule.


LogicGem - [Worksheet - C:\Users\Public\Documents\LogicGem\Examples\Shipping.lgt]
File Edit Table Logic Compiler Window Help
Shipping.lgt
Rule:

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
C1	Favored customer	y	y	y	y	n	n	n	n	n	n	n	n	n									
C2	Total order amount > $500.00	-	-	-	-	y	y	y	y	n	n	n	n	n									
C3	Total order weight > 25 kg	-	-	-	-	-	-	-	-	n	-	y	-	-									
C4	Shipping destination inside U.S.	y	y	n	n	y	y	n	n	y	y	y	n	n									
C5	Order requires special HAZMAT	y	n	y	n	n	y	y	n	n	y	n	y	n									
	*																						
A1	Apply 10% shipping discount					3	3	4	4														
A2	Free shipping	2	2	2	2																		
A3	Use FEDEX	1	1			1	1			1	1	1											
A4	Use Airborne Express			1	1			1	1				1	1									
A5	Phone customer for special handling	3		4			4	6			4		6										
A6	Special customs documentation			3	3			5	5				5	4									
A7	Shipping fee = total order weight *											2	2	2									
A8	Shipping fee = total order weight *					2	2	2	2	2	2												
A9	Add HAZMAT handling fee $45.00	4		5			5	7			3		4										
A10	Add foreign ship fee $17.50							3	3				3	3									
	*																						
	Frequency:	5	10	3	8	4	6	2	7	18	9	12	5	11									
	Cost:																						

Condition Stub: 1 | Completion Ratio: 32 : 32

Figure 31

Note also that we filled in the frequency row values at the bottom of the decision table with values showing the expected relative frequency of the rules. LogicGem provides an option for sorting rules in the resulting compilation based on frequency. Using this feature, the most frequently fired rules will be at the top of your code segment, resulting in the performance of your code being automatically optimized.

Frequency: You're able to enter a two digit number in the frequency row indicating an estimate of how often a particular rule might be expected to be invoked.

Now let’s have LogicGem generate our business logic by using the logic compiler to convert the decision table to English language rules. Start by selecting the **Compiler** | **Language** menu and select “English” and then selecting **Compiler** | **Structure** | **Nested If/Else**. Lastly, select **Compiler** | **Options** menu and turn on **Rule Numbers** and **Rule Descriptions** (turn on the right hand side check boxes).

Now select **Compiler** | **Compile** to tell LogicGem to compile the decision table in order to yield business rules. See the English business rules below. It is a useful exercise to trace through the English business rules and compare them with the decision table rules. By performing this manual check on a couple of rules, you’ll better understand how the decision table is translated by LogicGem into a logically complete structure.

```
Procedure: shipping.lgt

    If Favored customer is true, then
        If Shipping destination inside U.S. is true, then
            If Order requires special HAZMAT (hazardous materials)
                packing is true, then
                rule 1
                Favored USA customer with HAZMAT: FEDEX, free
                    shipping, phone, and HAZMAT fee
                Use FEDEX,
                Free shipping,
                Phone customer for special handling,
                Add HAZMAT handling fee $45.00,
            Otherwise,
                rule 2
                Favored USA customer, no HAZMAT: FEDEX and free
                    shipping
                Use FEDEX,
                Free shipping,
        Otherwise,
            If Order requires special HAZMAT (hazardous materials)
                packing is true, then
                rule 3
                Favored Int'l customer with HAZMAT:  Airborne,
                    free shipping, customs, phone, HAZMAT fee
                Use Airborne Express,
                Free shipping,
                Special customs documentation,
                Phone customer for special handling,
                Add HAZMAT handling fee $45.00,
            Otherwise,
                rule 4
                Favored Int'l customer, no HAZMAT: Airborne,
                    free shipping, and customs
                Use Airborne Express,
                Free shipping,
                Special customs documentation,
    Otherwise,
        If Total order amount > $500.00 is true, then
            If Shipping destination inside U.S. is true, then
                If Order requires special HAZMAT (hazardous materials)
                    packing is not true, then
                    rule 5
                    Big USA orders, of any weight, no HAZMAT:
                        FEDEX, lower rate and 10% discount
                    Use FEDEX,
                    Shipping fee = total order weight * $1.25/kg,
                    Apply 10% shipping discount,
                Otherwise,
                    rule 6
                    Big USA orders of any weight, with HAZMAT:
                        FEDEX, lower rate,10% discount, phone, and HAZMAT fee
                    Use FEDEX,
                    Shipping fee = total order weight * $1.25/kg,
                    Apply 10% shipping discount,
                    Phone customer for special handling,
                    Add HAZMAT handling fee $45.00,
            Otherwise,
                If Order requires special HAZMAT (hazardous materials)
                    packing is true, then
                    rule 7
                    Big Int'l orders of any weight, with HAZMAT:
                        Airborne, lower rate, foreign ship fee, 10% discount,
                        customs, phone, and HAZMAT fee
                    Use Airborne Express,
                    Shipping fee = total order weight * $1.25/kg,
                    Add foreign ship fee $17.50,
                    Apply 10% shipping discount,
                    Special customs documentation,
                    Phone customer for special handling,
                    Add HAZMAT handling fee $45.00,
                Otherwise,
                    rule 8
                    Big Int'l orders of any weight, no HAZMAT:
                        Airborne, lower rate, foreign ship fee, 10% discount,
                        and customs
                    Use Airborne Express,
                    Shipping fee = total order weight * $1.25/kg,
                    Add foreign ship fee $17.50,
                    Apply 10% shipping discount,
                    Special customs documentation,
        Otherwise,
            If Total order weight > 25 kg is not true, then
                If Shipping destination inside U.S. is true, then
                    If Order requires special HAZMAT (hazardous
                        materials) packing is not true, then
                        rule 9
                        Small USA orders with low weight, no
                            HAZMAT: FEDEX, and lower rate
                        Use FEDEX,
                        Shipping fee = total order weight * $1.25/kg,
                    Otherwise,
                        rule 10
                        Small USA orders, of any weight with
                            HAZMAT: FEDEX, lower rate, HAZMAT fee, and phone
                        Use FEDEX,
                        Shipping fee = total order weight * $1.25/kg,
                        Add HAZMAT handling fee $45.00,
                        Phone customer for special handling,
                Otherwise,
                    If Order requires special HAZMAT (hazardous
                        materials) packing is true, then
                        rule 12
                        Small Int'l orders with HAZMAT of any
                            weight: Airborne, higher rate, foreign fee,
                            HAZMAT fee, customs, and phone
                        Use Airborne Express,
                        Shipping fee = total order weight * $1.50/kg,
                        Add foreign ship fee $17.50,
                        Add HAZMAT handling fee $45.00,
                        Special customs documentation,
                        Phone customer for special handling,
                    Otherwise,
                        rule 13
                        Small Int'l orders of any weight, no
                            HAZMAT: Airborne, higher rate, foreign fee
                            and customs
                        Use Airborne Express,
                        Shipping fee = total order weight * $1.50/kg,
                        Add foreign ship fee $17.50,
                        Special customs documentation,
            Otherwise,
                If Shipping destination inside U.S. is true, then
                    If Order requires special HAZMAT (hazardous
                        materials) packing is true, then
                        rule 10
                        Small USA orders, of any weight with
                            HAZMAT: FEDEX, lower rate, HAZMAT fee, and phone
                        Use FEDEX,
                        Shipping fee = total order weight * $1.25/kg,
                        Add HAZMAT handling fee $45.00,
                        Phone customer for special handling,
                    Otherwise,
                        rule 11
                        Small USA order, high weight and no HAZMAT:
                            FEDEX at higher rate
                        Use FEDEX,
                        Shipping fee = total order weight * $1.50/kg,
                Otherwise,
                    If Order requires special HAZMAT (hazardous
                        materials) packing is true, then
                        rule 12
                        Small Int'l orders with HAZMAT of any
                            weight: Airborne, higher rate, foreign fee,
                            HAZMAT fee, customs, and phone
                        Use Airborne Express,
                        Shipping fee = total order weight * $1.50/kg,
                        Add foreign ship fee $17.50,
                        Add HAZMAT handling fee $45.00,
                        Special customs documentation,
                        Phone customer for special handling,
                    Otherwise,
                        rule 13
                        Small Int'l orders of any weight, no
                            HAZMAT: Airborne, higher rate, foreign fee
                            and customs
                        Use Airborne Express,
                        Shipping fee = total order weight * $1.50/kg,
                        Add foreign ship fee $17.50,
                        Special customs documentation,
End
```

Programming Examples

This section provides two examples of how to use LogicGem for the solution of programming problems. In each case, we'll start with a description of the business requirements and then demonstrate how to incrementally build a decision table to represent the logic. Once the table is logically complete, we'll finish up by generating the programming code that can be integrated with a software application.

Example 3: Sales Discount Calculation

The first programming example illustrates the use of a decision table to represent the programming logic for calculating a sales discount. The discount is to be based on two business requirements. First, a special discount is assigned if it is a new customer making the purchase. Second, a discount schedule is applied for existing customers based on the sales amount of the order, where a bigger discount is given for larger orders. Table 8 shows the discount rate schedule provided by the sales department of the company.

Business Rule	**Discount**
New customer	5%
Amount of sale less than $500.00	0%
Amount of sale $500.00 - $1,499.99	1%
Amount of sale $1,500.00 - $4,999.99	2%
Amount of sale greater than or equal to $5,000.00	5%

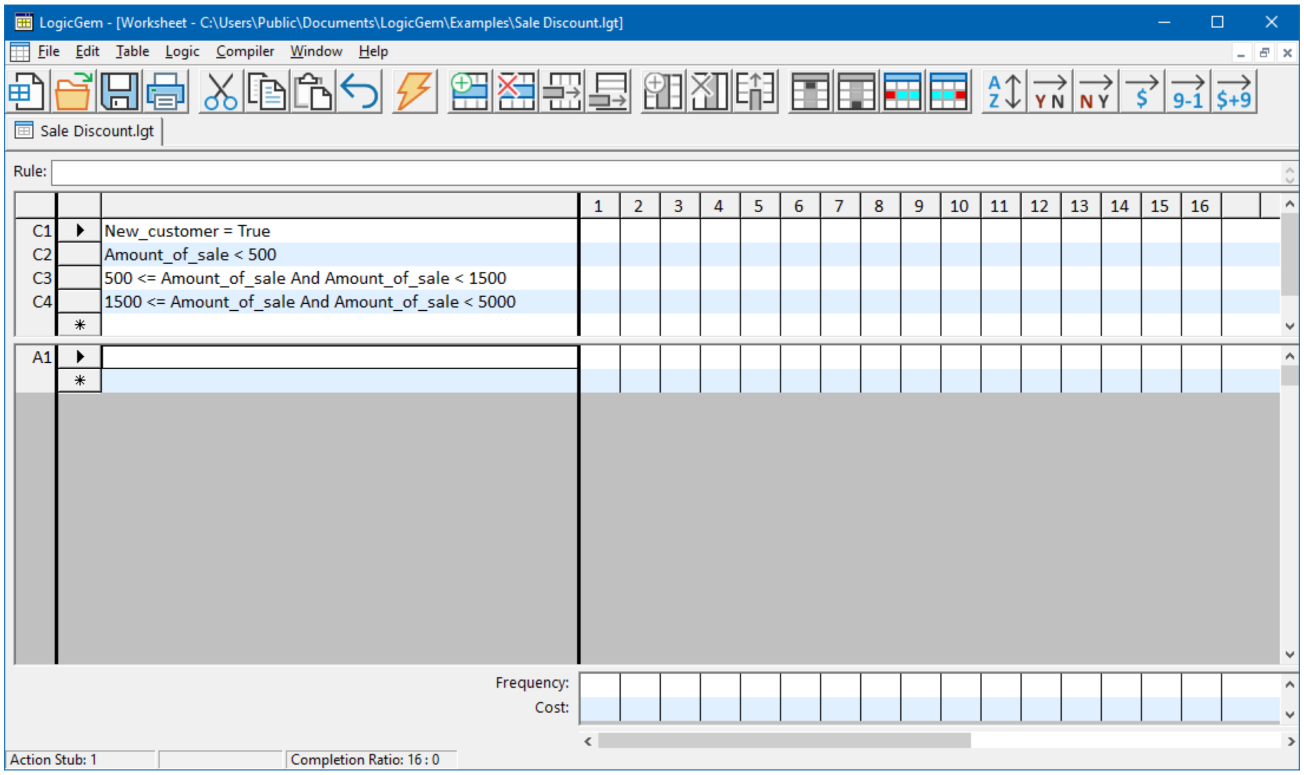
Table 8 – Sample Discount Rate Schedule

Our job as a programmer is to code this discount schedule in the programming language of choice, and make sure that all conditions have been considered. Typically, developers will attempt to convert business rules such as this without any software engineering tools. The resulting code for simple requirements (such as this example) has a high probability of success, however for complex sets of requirements it becomes increasingly likely that some rules will be missed, resulting in incomplete program logic, and fallible software applications.

Let's use Logic Gem to develop the required program logic in a way that we're 100% confident that the code will be complete.

You can load a completed version of this table by selecting **File | Open**, highlighting the sale_discount.lgt table file in the LogicGem Examples folder, and clicking the Open button. Optionally, you can begin experiencing how easy LogicGem is to use by actually building this next example from scratch.

We'll start by opening up a new decision table by selecting **File | New**. The first thing we'll do is fill in all the condition stubs we can think of. We can obtain the list of conditions from the discount rate schedule table and enter them as condition stubs. Since this is a programming example, the condition stubs must be valid logical expressions of the target programming language (in our case Visual Basic .NET). Figure 32 shows four condition stubs: C1, C2, C3, and C4.


LogicGem - [Worksheet - C:\Users\Public\Documents\LogicGem\Examples\Sale Discount.lgt]
File Edit Table Logic Compiler Window Help
Sale Discount.lgt
Rule:
Frequency:
Cost:
Action Stub: 1
Completion Ratio: 16 : 0

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
C1 New_customer = True																
C2 Amount_of_sale < 500																
C3 500 <= Amount_of_sale And Amount_of_sale < 1500																
C4 1500 <= Amount_of_sale And Amount_of_sale < 5000																
A1																

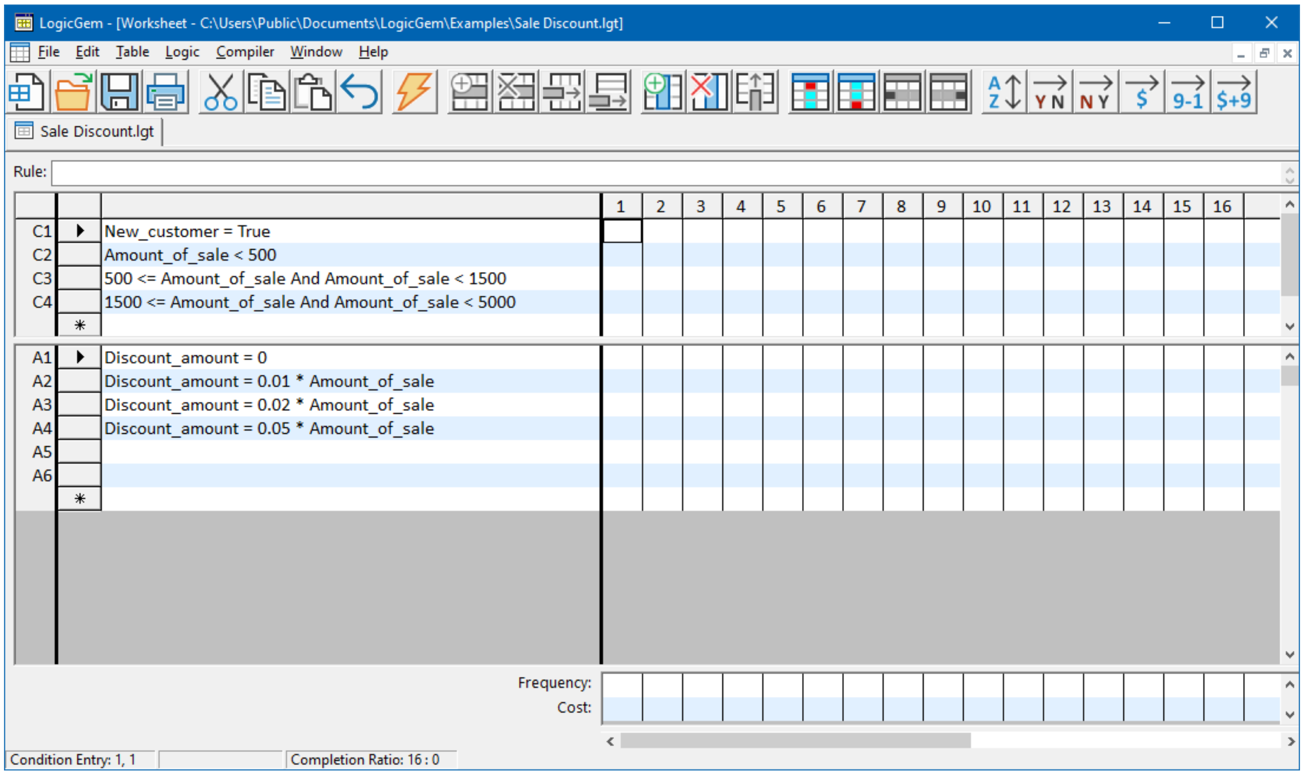
Figure 32

You might wonder what about the last rule in the discount rate schedule table? We can add the last rule which accounts for large orders over $5,000, but if you consider how a decision table works, you’ll see that we can get away with eliminating this rule. We’ll return to this issue later when we fill in the rules.

Next, let’s fill in all the action stubs we can come up with. We can base the action stubs on the above discount table as well. Specifically, we can use the discount percentages in the table to formulate the actions. Since we wish for Logic Gem to generate actual programming code for this example, we must code the action stubs in the appropriate programming language. In this example we’ll use Visual Basic .NET to code the stubs, and the stubs shall contain simple assignment statements.

We see that the action stubs are simple programming assignment statements that calculate the discount amount based on the variable “Amount_of_sale” and then assign it to another variable “Discount_amount”. So based on the business requirements, we come up with action stubs: A1, A2, A3, and A4 as shown in Figure 33.

Notice that the Completion Ratio appearing in the status bar of the screen shows "16:0" meaning that based on the conditions you've entered, there are 16 rules that must be accounted for. Currently no rules are defined so a 0 is shown as the number accounted for.


LogicGem - [Worksheet - C:\Users\Public\Documents\LogicGem\Examples\Sale Discount.lgt]
File Edit Table Logic Compiler Window Help
Sale Discount.lgt
Rule:
Frequency:
Cost:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
C1 New_customer = True																
C2 Amount_of_sale < 500																
C3 500 <= Amount_of_sale And Amount_of_sale < 1500																
C4 1500 <= Amount_of_sale And Amount_of_sale < 5000																
A1 Discount_amount = 0																
A2 Discount_amount = 0.01 * Amount_of_sale																
A3 Discount_amount = 0.02 * Amount_of_sale																
A4 Discount_amount = 0.05 * Amount_of_sale																
A5																
A6																

Condition Entry: 1, 1 Completion Ratio: 16 : 0

Figure 33

The next step in building the decision table is to start creating rules based on the condition and action stubs. Let's start with an easy one. Our business requirements state that all new customers should receive a 5% discount on their first purchase. We'll assume there is a Boolean variable available in the code base called "New_customer" which has a True value if the customer is new. With this understanding, let's place a "Y" in the Rule 1 column for condition C1, and place a don't-care "-" symbol in the Rule 1 column for conditions C2, C3, and C4. This says that if the customer is new, it doesn't matter what the amount of sale is to get the discount.

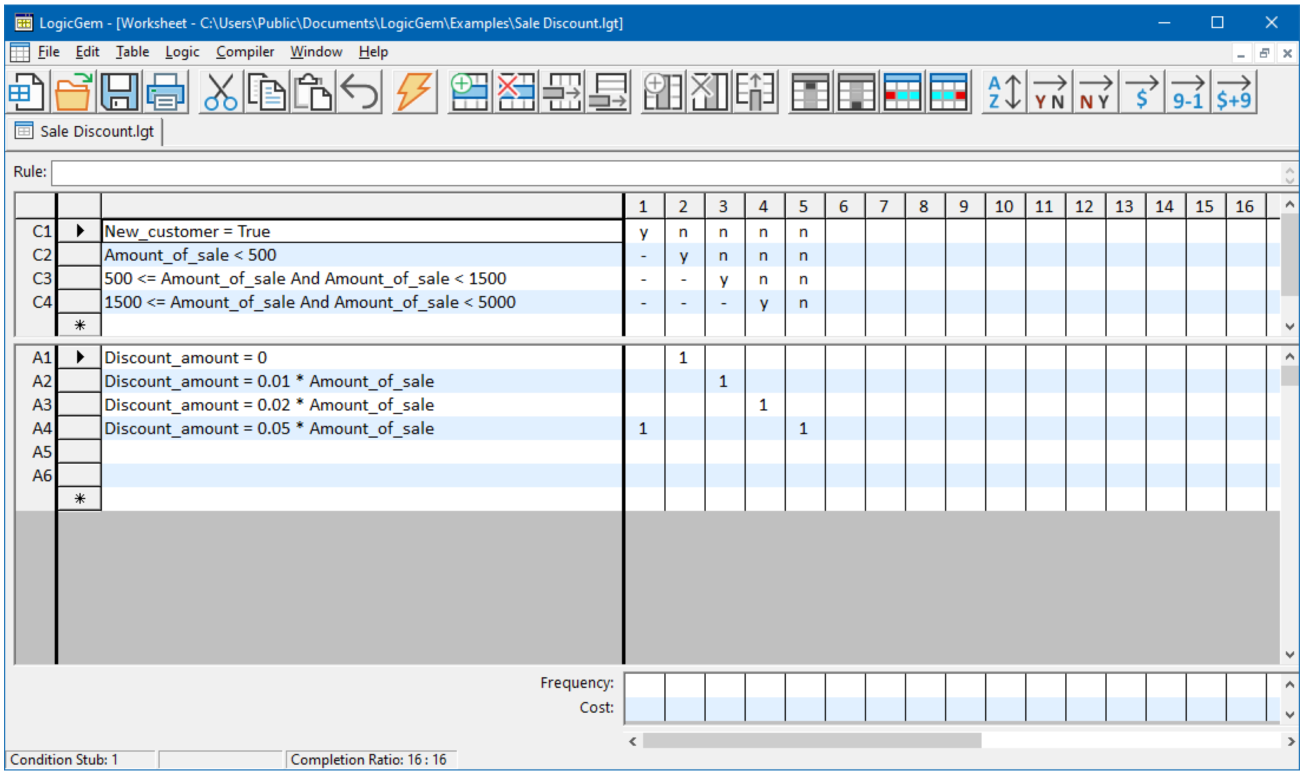
Now, let's place a "1" in the Rule 1 column for action A4. Coupled with the condition part of Rule 1, this rule says to calculate a 5% discount amount if the customer is new. You can document the rule in the decision table by entering a rule description in the Rule textbox: "New customers get highest discount as incentive." When LogicGem generates code, you'll have the option of including rule descriptions.

We can add another rule to represent a small order of less than $500 by entering "Y" in the Rule 2 column for condition C2, and "N" for both C3 and C4. We also need to place "N" in C1 since this order is not for a new customer.

Rule 3 is defined in a similar way, "N" for C1, C2, and C4, and "Y" for C3. For Rule 4, enter "N" for C1, C2, and C3, and "Y" for C4.

The final rule is Rule 5 and is handled in an interesting way. This rule satisfies the requirement for large orders over $5,000. We do this by entering "N" for C1, C2, C3, and C4. The business requirement for "Amount of sale greater than or equal to $5,000.00” is handled by a process of elimination since if conditions C2, and C3, and C4 are all false, then we reference action A4 in the action stub.

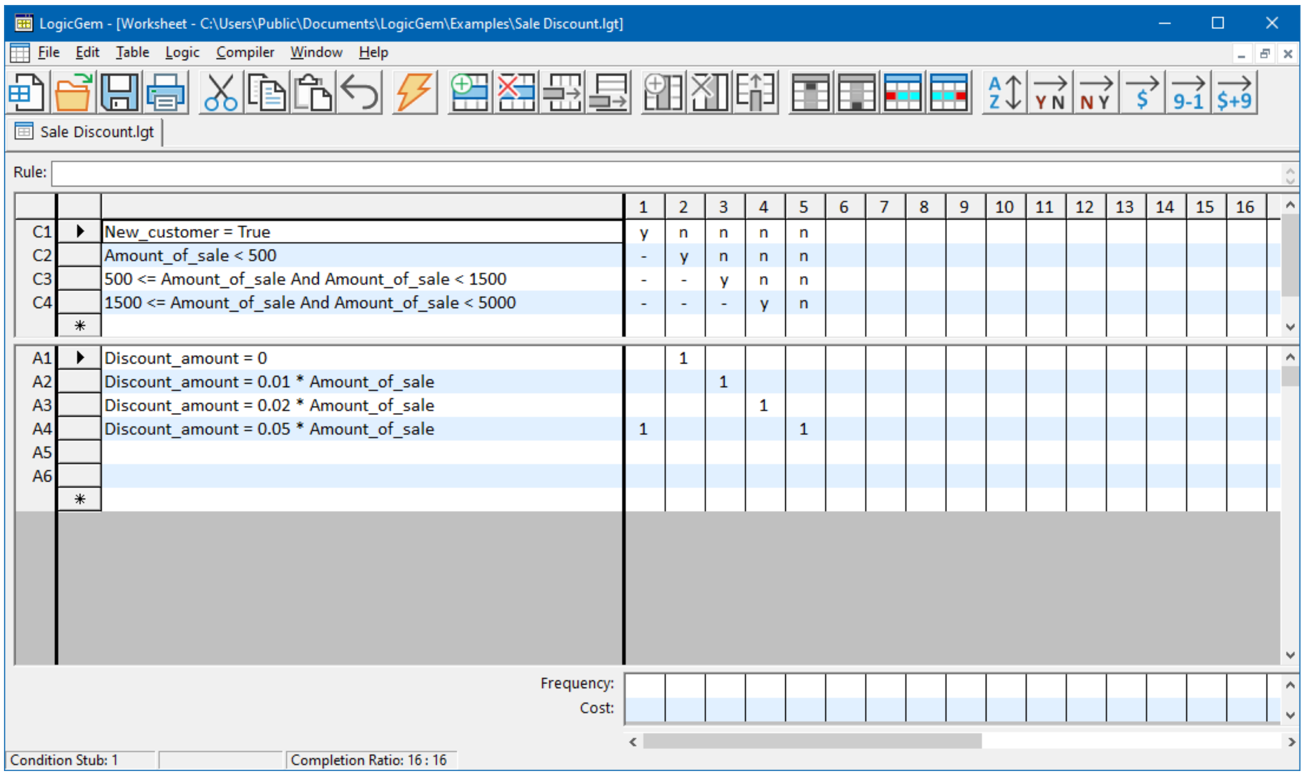
The completed table with the rules representing all the business requirements is shown in Figure 34. But notice that the Completion Ratio is 16:12 which means the table is logically incomplete. We need to reconsider the rules we’ve defined thus far to see if there is a way to reconstruct any of them to make the table complete.


LogicGem - [Worksheet - C:\Users\Public\Documents\LogicGem\Examples\Sale Discount.lgt]
File Edit Table Logic Compiler Window Help
Sale Discount.lgt
Rule:
Frequency:
Cost:
Condition Stub: 1
Completion Ratio: 16 : 16

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
C1	New_customer = True	y	n	n	n	n											
C2	Amount_of_sale < 500	-	y	n	n	n											
C3	500 <= Amount_of_sale And Amount_of_sale < 1500	-	-	y	n	n											
C4	1500 <= Amount_of_sale And Amount_of_sale < 5000	-	-	-	y	n											
A1	Discount_amount = 0		1														
A2	Discount_amount = 0.01 * Amount_of_sale			1													
A3	Discount_amount = 0.02 * Amount_of_sale				1												
A4	Discount_amount = 0.05 * Amount_of_sale	1				1											
A5																	
A6																	

Figure 34

We can modify Rule 2 by entering don’t care values for C3 and C4 since if C2 is true, we don’t really care what C3 and C4 are. The action for Rule 2 should be A1 set to 1. We can continue with Rule 3 by entering a don’t care value for C4. The action for Rule 3 should be A2 set to 1. Rule 4 may be left as-is, with the action A3 set to 1. Now the more optimized table in Figure 35 shows a Completion Ratio of 16:16 indicating a logically complete decision table. For good measure you can select the Logic | Ambiguity Check in order to check the robustness of the decision table. If there are no ambiguous rules in the table, LogicGem will report “Table is unambiguous.” If the table is ambiguous, you can select the **Logic** | **Disambiguate** menu option, or the **Logic** | **Expand** option to continue building the logic.


LogicGem - [Worksheet - C:\Users\Public\Documents\LogicGem\Examples\Sale Discount.lgt]

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
C1	New_customer = True	y	n	n	n	n											
C2	Amount_of_sale < 500	-	y	n	n	n											
C3	500 <= Amount_of_sale And Amount_of_sale < 1500	-	-	y	n	n											
C4	1500 <= Amount_of_sale And Amount_of_sale < 5000	-	-	-	y	n											
A1	Discount_amount = 0		1														
A2	Discount_amount = 0.01 * Amount_of_sale			1													
A3	Discount_amount = 0.02 * Amount_of_sale				1												
A4	Discount_amount = 0.05 * Amount_of_sale	1				1											
A5																	
A6																	

Figure 35

Our final task is to configure Logic Gem’s logic compiler to convert the decision table to programming code. Start by selecting the **Compiler** | **Language** menu and select “Visual Basic .NET” and then selecting **Compiler** | **Structure** | **Nested If/Else**. Lastly, select **Compiler** | **Options** menu and turn on **Rule Numbers** and **Rule Descriptions** (turn on the right hand side check boxes).

Now select **Compiler** | **Compile** to tell LogicGem to compile the decision table in order to yield programming code. See the Visual Basic .NET code below.

```
' sale_discount.lgt

If (New_customer = True) Then
    ' rule 1
    ' New customers get highest discount as incentive.
    Discount_amount = 0.05 * Amount_of_sale
Else
    If (Amount_of_sale < 500) Then
        ' rule 2
        ' Small sales get no discount.
        Discount_amount = 0
    Else
        If (500 <= Amount_of_sale And Amount_of_sale < 1500) Then
            ' rule 3
            ' Mid-level sales amount gets 1% discount.
            Discount_amount = 0.01 * Amount_of_sale
        Else
```

```
            If (1500 <= Amount_of_sale And Amount_of_sale < 5000) Then
                ' rule 4
                ' Higher sales amount gets 2% discount.
                Discount_amount = 0.02 * Amount_of_sale
            Else
                ' rule 5
                ' Large sales, over $5,000 get 5% discount.
                Discount_amount = 0.05 * Amount_of_sale
            End If
        End If
    End If
End If
```

The resulting VB .NET code easily can be merged with a main program in order to calculate the sales discount. Of course program variables used in the logic code must be declared in the main program.

Example 4: Dynamically Build SQL Statements

The final example use of LogicGem illustrates how decision tables can be useful in generating programming code to dynamically build structured query language (SQL) statements. Quite often, programmers are faced with accessing databases in a manner that is dictated by a complex set of business rules. The resulting SQL SELECT statement, for example, might access a specific table in a database (or tables via a relational join), select a specific field list, and retrieve records based on a series of selection criteria. In the case we'll consider in this section, the decision table will drive the process of defining logic to build the SQL WHERE clause which is often the most complex part of the SELECT statement.

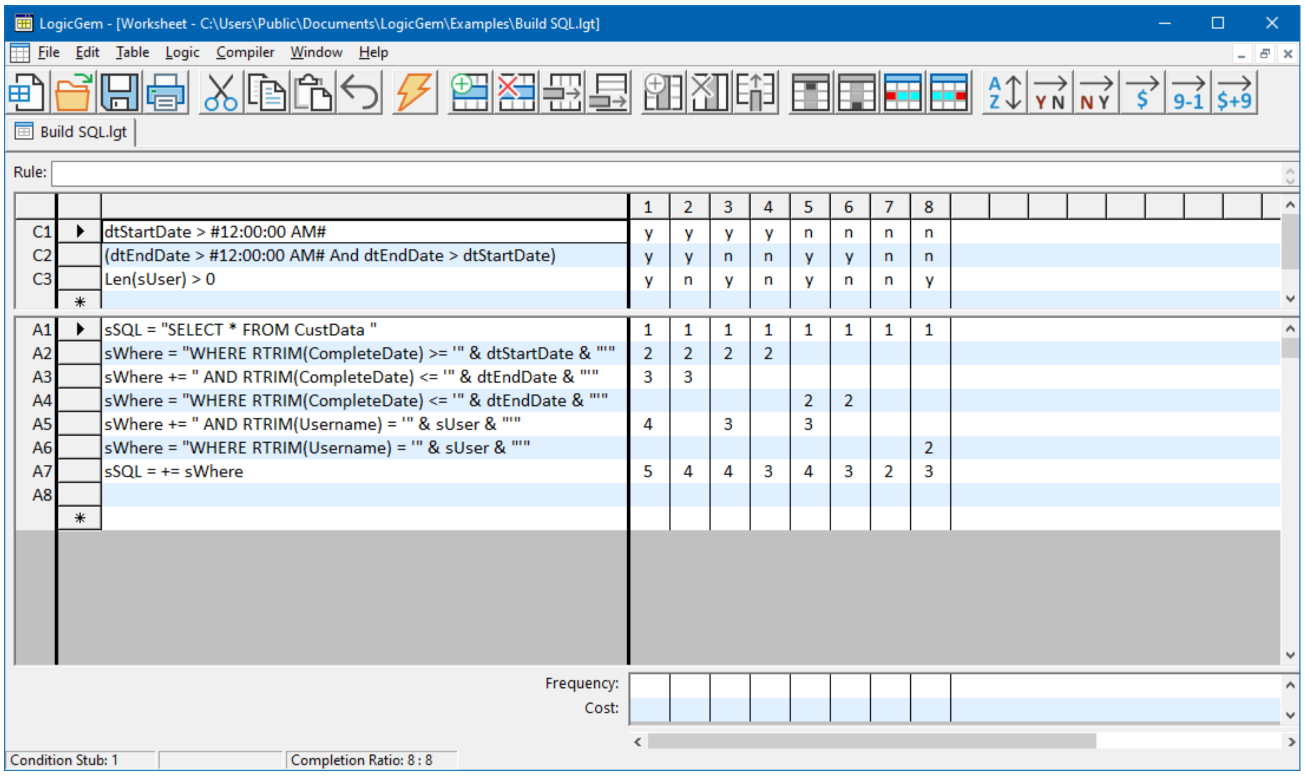
The business requirement is to pull data from a customer data table in a relational database. The table has two fields that will participate in the selection criteria: CompleteDate, and Username. The values used for comparison with the field values are held in three program variables: dtStartDate, dtEndDate, and sUser. The program also defines two additional variables: sSQL and sWhere to store the constructed SQL statement.

One of the conditions the decision table needs to consider is whether the WHERE clause includes a range of date values for the field CompleteDate, or just the upper date range. It must also consider the case where no Username field value is provided.

Review the decision table in Figure 36 that includes all the conditions, actions, and rules for this example. You can load a completed version of this table by selecting **File** | **Open**, highlighting the build_sql.lgt table file in the LogicGem Examples folder, and clicking the Open button.

The conditions and actions are shown using Visual Basic .NET syntax. As an example, let's take a closer look at Rule 1 which accounts for the case where there is both an upper and lower value for CompleteDate, and there is a value for Username. In this case, the following code will be generated by the decision table based on five different actions included in Rule 1.

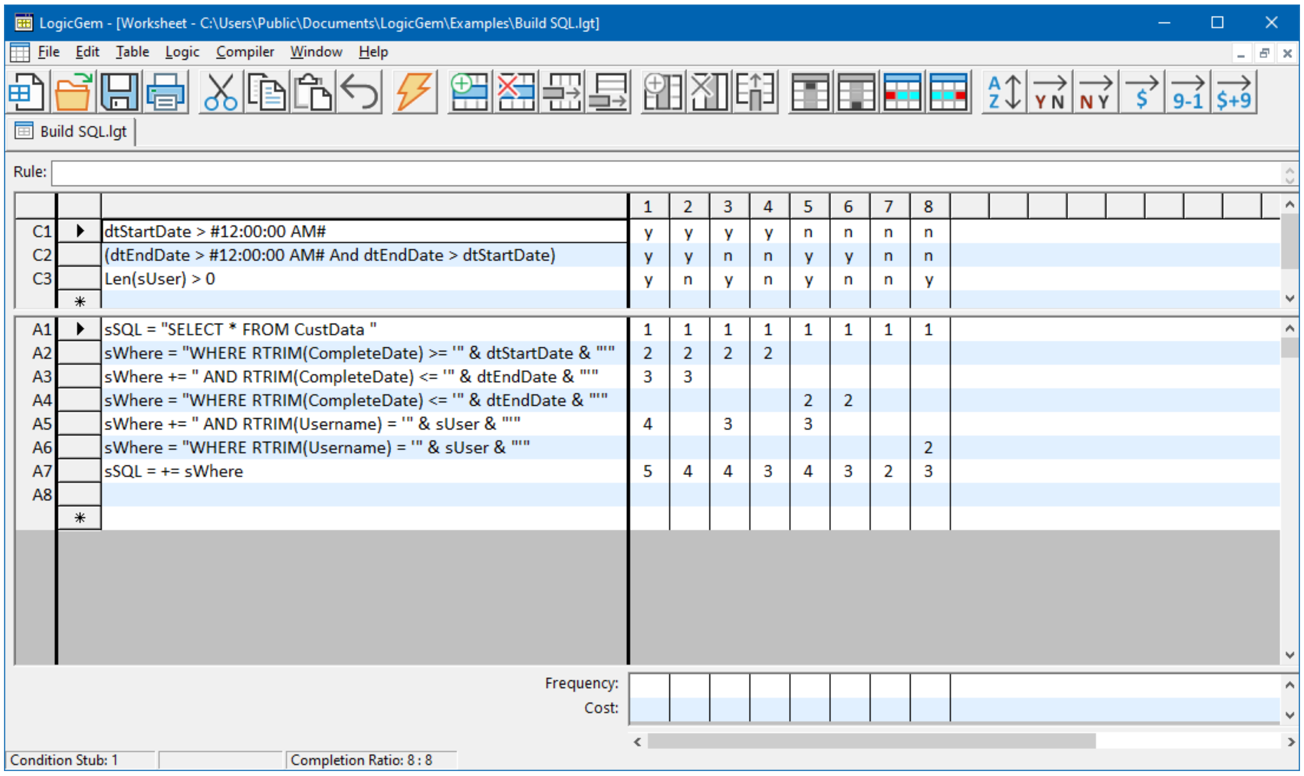
```
sSQL = "SELECT * FROM CustData "
         sWhere = "WHERE RTRIM(CompleteDate) >= '" & dtStartDate & "'"
         sWhere += " AND RTRIM(CompleteDate) <= '" & dtEndDate & "'"
         sWhere += " AND RTRIM(Username) = '" & sUser & "'"
         sSQL = += sWhere
```


LogicGem - [Worksheet - C:\Users\Public\Documents\LogicGem\Examples\Build SQL.lgt]
File Edit Table Logic Compiler Window Help
Build SQL.lgt
Rule:

		1	2	3	4	5	6	7	8
C1	dtStartDate > #12:00:00 AM#	y	y	y	y	n	n	n	n
C2	(dtEndDate > #12:00:00 AM# And dtEndDate > dtStartDate)	y	y	n	n	y	y	n	n
C3	Len(sUser) > 0	y	n	y	n	y	n	n	y
A1	sSQL = "SELECT * FROM CustData "	1	1	1	1	1	1	1	1
A2	sWhere = "WHERE RTRIM(CompleteDate) >= '" & dtStartDate & "'"	2	2	2	2				
A3	sWhere += " AND RTRIM(CompleteDate) <= '" & dtEndDate & "'"	3	3						
A4	sWhere = "WHERE RTRIM(CompleteDate) <= '" & dtEndDate & "'"					2	2		
A5	sWhere += " AND RTRIM(Username) = '" & sUser & "'"	4		3		3			
A6	sWhere = "WHERE RTRIM(Username) = '" & sUser & "'"								2
A7	sSQL = += sWhere	5	4	4	3	4	3	2	3
A8									

Figure 36

But there is a deficiency in the table because the Completion Ratio is 8:7 which means we're missing a rule. As is often the case (and the primary reason for using a logic tool like LogicGem) we need help to find the missing rule. Fortunately, we can use a tool within LogicGem to come up with the rule. Select the Missing tool from the Logic menu, and the missing Rule 8 is determined. You can then proceed to enter in the appropriate actions. See Figure 37 for the completed decision table.



LogicGem - [Worksheet - C:\Users\Public\Documents\LogicGem\Examples\Build SQL.lgt]

Build SQL.lgt

Rule:

		1	2	3	4	5	6	7	8
C1	dtStartDate > #12:00:00 AM#	y	y	y	y	n	n	n	n
C2	(dtEndDate > #12:00:00 AM# And dtEndDate > dtStartDate)	y	y	n	n	y	y	n	n
C3	Len(sUser) > 0	y	n	y	n	y	n	n	y
A1	sSQL = "SELECT * FROM CustData "	1	1	1	1	1	1	1	1
A2	sWhere = "WHERE RTRIM(CompleteDate) >= '" & dtStartDate & "'"	2	2	2	2				
A3	sWhere += " AND RTRIM(CompleteDate) <= '" & dtEndDate & "'"	3	3						
A4	sWhere = "WHERE RTRIM(CompleteDate) <= '" & dtEndDate & "'"					2	2		
A5	sWhere += " AND RTRIM(Username) = '" & sUser & "'"	4		3		3			
A6	sWhere = "WHERE RTRIM(Username) = '" & sUser & "'"								2
A7	sSQL = += sWhere	5	4	4	3	4	3	2	3
A8									

Frequency:
Cost:

Condition Stub: 1 | Completion Ratio: 8 : 8

Figure 37

The last thing we need to do is to configure Logic Gem's logic compiler to convert the decision table to programming code. Select the Compiler | Language menu and chose "Visual Basic .NET" and then selecting **Compiler** | **Structure** | **Nested If/Else**.

Now select **Compiler** | **Compile** to tell LogicGem to compile the decision table in order to yield programming code. See the Visual Basic .NET code below. A programmer with reasonable talent could probably handle writing the code by hand, but if we add five more conditions and ten additional actions, the task becomes much more daunting. LogicGem can handle a higher degree of complexity than a human software engineer, all with a 100% level of completeness.

```
' build_sql_2.lgt

If (dtStartDate > #12:00:00 AM#) Then
    If ((dtEndDate > #12:00:00 AM# And dtEndDate > dtStartDate)) Then
        If (Len(sUser) > 0) Then
            ' rule 1
            sSQL = "SELECT * FROM CustData "
            sWhere = "WHERE RTRIM(CompleteDate) >= '" & dtStartDate & "'"
            sWhere += " AND RTRIM(CompleteDate) <= '" & dtEndDate & "'"
            sWhere += " AND RTRIM(Username) = '" & sUser & "'"
            sSQL = += sWhere
        Else
            ' rule 2
            sSQL = "SELECT * FROM CustData "
            sWhere = "WHERE RTRIM(CompleteDate) >= '" & dtStartDate & "'"
            sWhere += " AND RTRIM(CompleteDate) <= '" & dtEndDate & "'"
            sSQL = += sWhere
        End If
    Else
        If (Len(sUser) > 0) Then
            ' rule 3
            sSQL = "SELECT * FROM CustData "
            sWhere = "WHERE RTRIM(CompleteDate) >= '" & dtStartDate & "'"
            sWhere += " AND RTRIM(Username) = '" & sUser & "'"
            sSQL = += sWhere
        Else
            ' rule 4
            sSQL = "SELECT * FROM CustData "
            sWhere = "WHERE RTRIM(CompleteDate) >= '" & dtStartDate & "'"
            sSQL = += sWhere
        End If
    End If
Else
    If ((dtEndDate > #12:00:00 AM# And dtEndDate > dtStartDate)) Then
        If (Len(sUser) > 0) Then
            ' rule 5
            sSQL = "SELECT * FROM CustData "
            sWhere = "WHERE RTRIM(CompleteDate) <= '" & dtEndDate & "'"
            sWhere += " AND RTRIM(Username) = '" & sUser & "'"
            sSQL = += sWhere
        Else
            ' rule 6
            sSQL = "SELECT * FROM CustData "
            sWhere = "WHERE RTRIM(CompleteDate) <= '" & dtEndDate & "'"
            sSQL = += sWhere
        End If
    Else
        If Not (Len(sUser) > 0) Then
            ' rule 7
            sSQL = "SELECT * FROM CustData "
            sSQL = += sWhere
        Else
            ' rule 8
            sSQL = "SELECT * FROM CustData "
            sWhere = "WHERE RTRIM(Username) = '" & sUser & "'"
            sSQL = += sWhere
        End If
    End If
End If
```

Preferences

Introduction

LogicGem is highly flexible in how it allows users to customize a variety of attributes that affect the operation of the program. LogicGem provides the Preferences dialog box as the means to set these attributes to suit a user's requirements. The Preferences dialog box is organized into three groups of attributes (distinct tabs): General, Compiler, and Appearance. The attribute settings are retained from session to session until they are changed.

You may select the Preferences dialog using the following method:

Menu bar

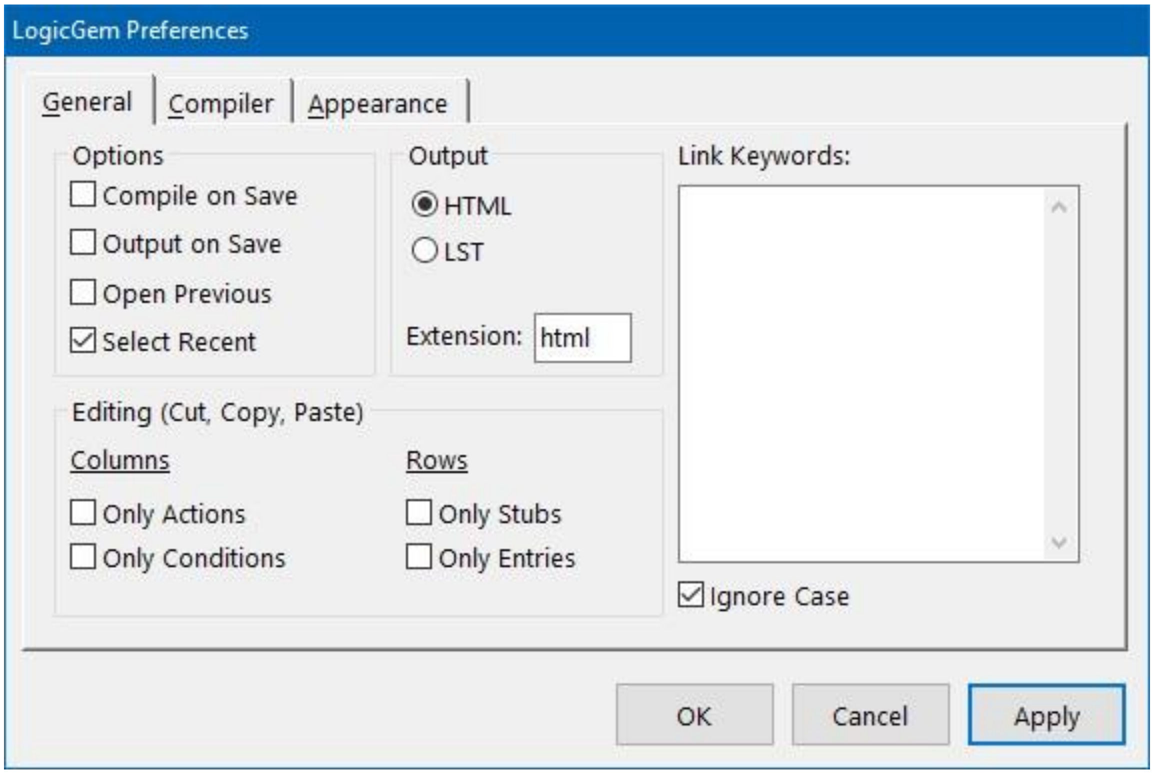
Click on the Edit menu function.

Click on the Preferences option.

Click on the desired tab: General, Compiler, or Appearance, or use the left/right arrow keys to move between tabs.

General Preferences

The General preferences dialog box shown in Figure 38 has default settings for the Save, Output and Edit functions. It also provides an option to determine if LogicGem will allow uppercase condition entries (Ignore Case on Entries).


LogicGem Preferences

General | Compiler | Appearance

Options
- [ ] Compile on Save
- [ ] Output on Save
- [ ] Open Previous
- [x] Select Recent

Output
- (●) HTML
- ( ) LST

Extension: html

Editing (Cut, Copy, Paste)

Columns
- [ ] Only Actions
- [ ] Only Conditions

Rows
- [ ] Only Stubs
- [ ] Only Entries

Link Keywords:

- [x] Ignore Case

OK | Cancel | Apply

*Figure 38 – General **Preferences** Dialog Box*

Save Defaults

The *Compile on Save* checkbox configures LogicGem to automatically produce a compiler output file each time you save the decision table. The compiler output file will honor all compiler settings such as language and verbosity level. The generated compiler output file has the same name as the decision table, with filename extension appropriate for the target language.

The *Output on Save* checkbox configures LogicGem to automatically produce an output file (HTML or LST text file) each time you save the decision table. The type of output file generated depends on the Output To setting discussed in the next section.

The *Save ALL CAPS* checkbox configures LogicGem to save each filename in uppercase.

Output To

The *Output To* preference setting is used in conjunction with the Output To option on the File menu, and the Output on Save preference setting described in the previous section. The Output To setting determines which file format to create. Use the option button to specify either “As HTML” or “As LST” along with the Extension textbox where you can enter the desired filename extension. For example, selecting “As HTML” causes the Output To function to create a Hypertext Markup Language (HTML) version of the logic table. You can open this file in your web browser to view the table. If you wanted to use a filename extension other than the default "HTML", you can enter it in the Extension textbox (e.g. "HTM").

Case Default

The *Ignore Case on Entries* checkbox determines whether LogicGem is to ignore or accept mixed case condition entries, i.e. "y" for Yes, "n" for No, "Y" for No, and "N" for Yes. Click on the checkbox to insert a checkmark and instruct LogicGem to translate all entries to lowercase. A blank checkbox instructs LogicGem to accept both upper and lowercase entries.

Reminder: Condition entries can be in upper or lowercase but they are interpreted differently:
Lowercase entries are interpreted as y = yes and n = no.
Uppercase entries are interpreted as Y = no and N = yes.

Link Keywords

The decision tables can reference one another using "hyperlinks." When a table refers to another table via a user-defined link keyword (e.g. "goto"), LogicGem generates a hyperlink that allows the user to navigate from one table to the other with a single click. Hyperlinks can appear in both the condition and action stubs.

LogicGem determines when to create a hyperlink using a list of keywords provided by the logic engineer (via the **Edit | Preferences** dialog box in the General tab). When one of the user-defined keyword is found in a stub, LogicGem automatically generates a hyperlink to the name following the keyword. For example, if GOTO is a link keyword you decide to use and an action stub value is set to

```
GOTO tablexyz
```

LogicGem creates a hyperlink to the table "tablexyz" which must exist in the current decision table directory.

This linking is available in both the generated HTML page and the LogicGem Worksheet. When viewing an HTML generated page using a browser, any decision table link appears just as a regular Web hyperlink. When clicking on a hyperlink in a LogicGem worksheet, the corresponding table is open in a new window. To edit a stub that contains a link, a right-click is necessary because the left-click is used to navigate.

Edit (Cut, Copy, Paste) Behavior

The edit default options provide a way to modify decision table components independently. Cut, copy or paste functions are performed only on the selected (check-marked) default options.

Perform Column Operations on… - Using this preference, column edits can be restricted by:

Clicking on the *Only Action Entries* check box to restrict edits to the Action entries

OR

Clicking on the *Only Condition Rules* check box to restrict edits to the Condition entries.

Perform Row Operations on … - Using this preference, row edits can be restricted by:

Clicking on the *Only Stubs* check box to restrict edits to the (condition and action) stub entries

OR

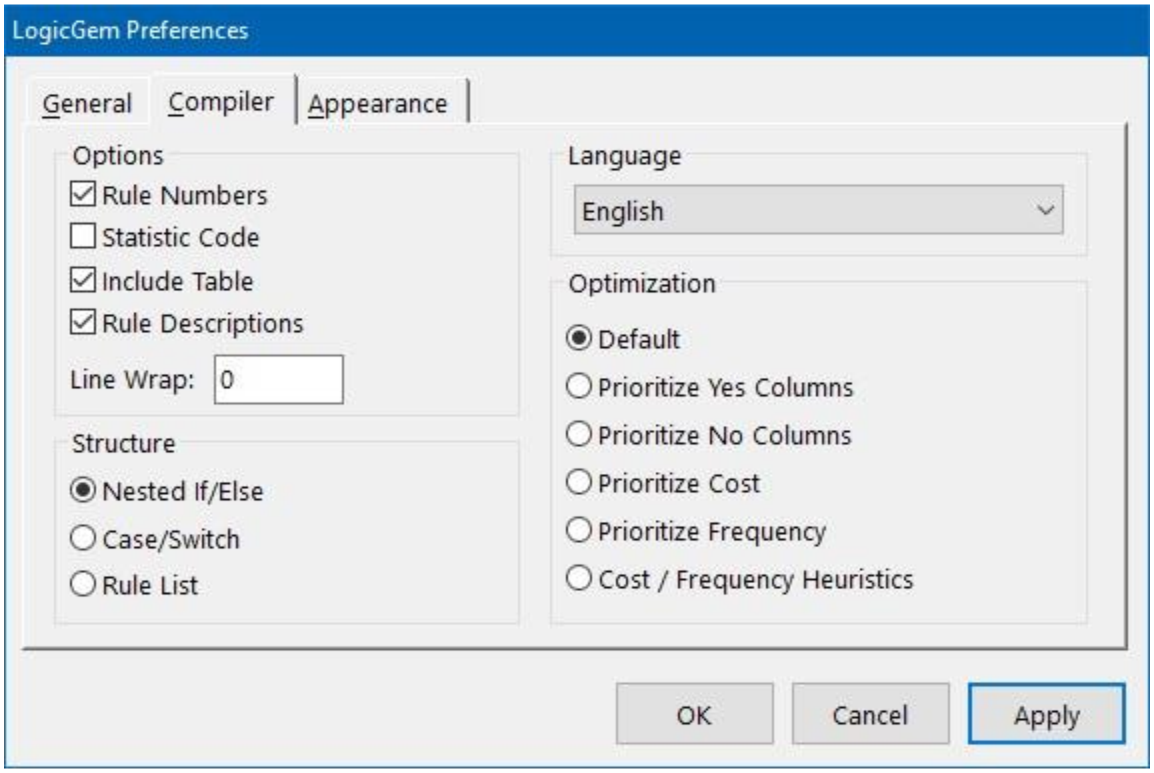
Clicking on the *Only Entries* check box to restrict edits to the (condition and action) entries.

Compiler Preferences

The Compiler preferences dialog box shown in Figure 39 has default settings for the Options, Structure, Language, and Sort functions.

Options

The Options section of the Compiler preferences dialog box allows you to specify the verbosity level for the LogicGem compiler. By checking the individual boxes, you can activate Rule Numbers, Statistic Code, Table, and Rule Descriptions in the compiled code. As many options as desired may be checked. For more information on each setting, see the *Compiling Decision Tables* chapter.



Options
[✓ Rule Numbers]
Statistic Code
[✓ Include Table]
[✓ Rule Descriptions]
Line Wrap: 0

Structure
[✓ Nested If/Else]
Case/Switch
Rule List

Language
English

Optimization
[✓ Default]
Prioritize Yes Columns
Prioritize No Columns
Prioritize Cost
Prioritize Frequency
Cost / Frequency Heuristics

Figure 39 – CompilerPreferences Dialog Box

Structure

The Structure section of the Compiler preferences dialog box allows you to specify the structure of the code generated by the LogicGem compiler. By clicking the desired option button, you can select a structure: Nested If/Else, Case/Switch, or Rule List. Only one structure may be selected. For more information on each structure, see the *Compiling Decision Tables* chapter.

Language

The Language pull-down list of the Compiler preferences dialog box allows you to select the target natural or programming language for the LogicGem compiler. Only one language may be specified at a time. For more information on each setting, see the *Compiling Decision Tables* chapter.

Optimization

The Optimization section of the Compiler preferences dialog box allows you to specify the optimization method used for rules in the code generated by the LogicGem compiler. By clicking the desired option button, you can select from a set of six options. Only one option may be specified at a time. For more information on each setting, see the *Compiling Decision Tables* chapter.

NOTE: It is recommended you use the default optimization in most cases. Due to its inherent nature, the Compiler sort option is generally only appropriate for the Nested If/Else structure.

Appearance Defaults

In order to suit personal stylistics preferences in the LogicGem Worksheet, the Appearance preferences dialog box shown in Figure 40 has default settings for the Color and Font for specific areas of the Worksheet.

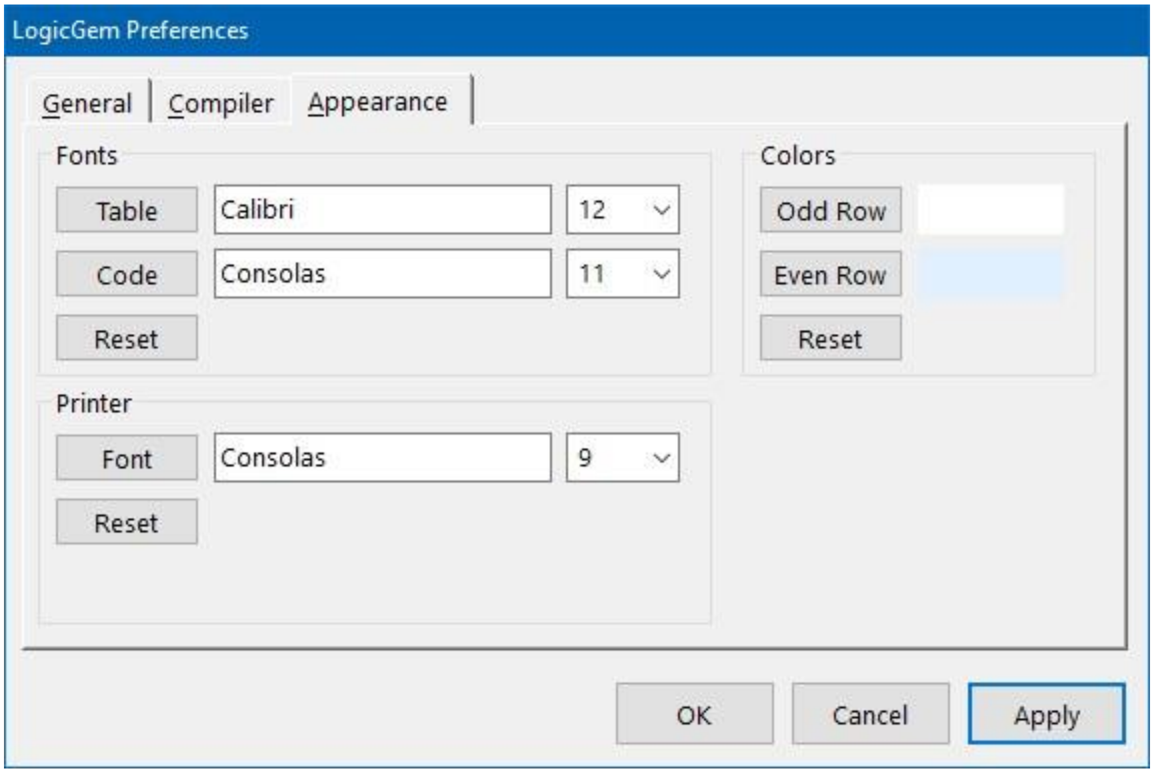
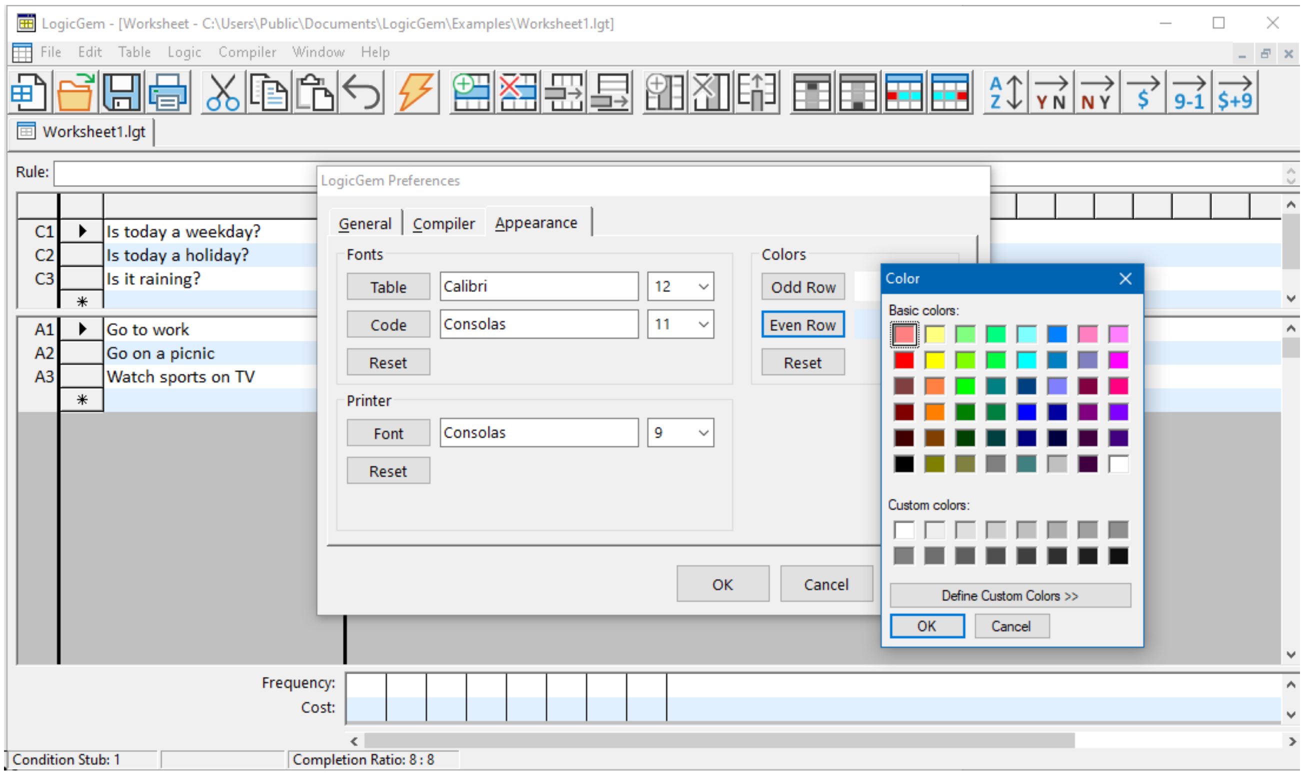


Table: Calibri 12
Code: Consolas 11
Printer Font: Consolas 9

Figure 40 – Appearance Preferences Dialog Box

Color

The Color section of the Appearance preferences dialog box allows you to specify the color theme for odd and/or even rows in the Worksheet. The default white color for odd rows, and light yellow color for even rows may be customized by using the special color palette as shown in the figure. By selecting the desired color, and clicking the Apply button, the Worksheet will be displayed in the new color theme.



LogicGem - [Worksheet - C:\Users\Public\Documents\LogicGem\Examples\Worksheet1.lgt]

Worksheet1.lgt

Rule:

C1 Is today a weekday?
C2 Is today a holiday?
C3 Is it raining?

A1 Go to work
A2 Go on a picnic
A3 Watch sports on TV

Frequency:
Cost:

Condition Stub: 1 | Completion Ratio: 8 : 8

LogicGem Preferences

Fonts

Table	Calibri	12
Code	Consolas	11

Printer

Font	Consolas	9

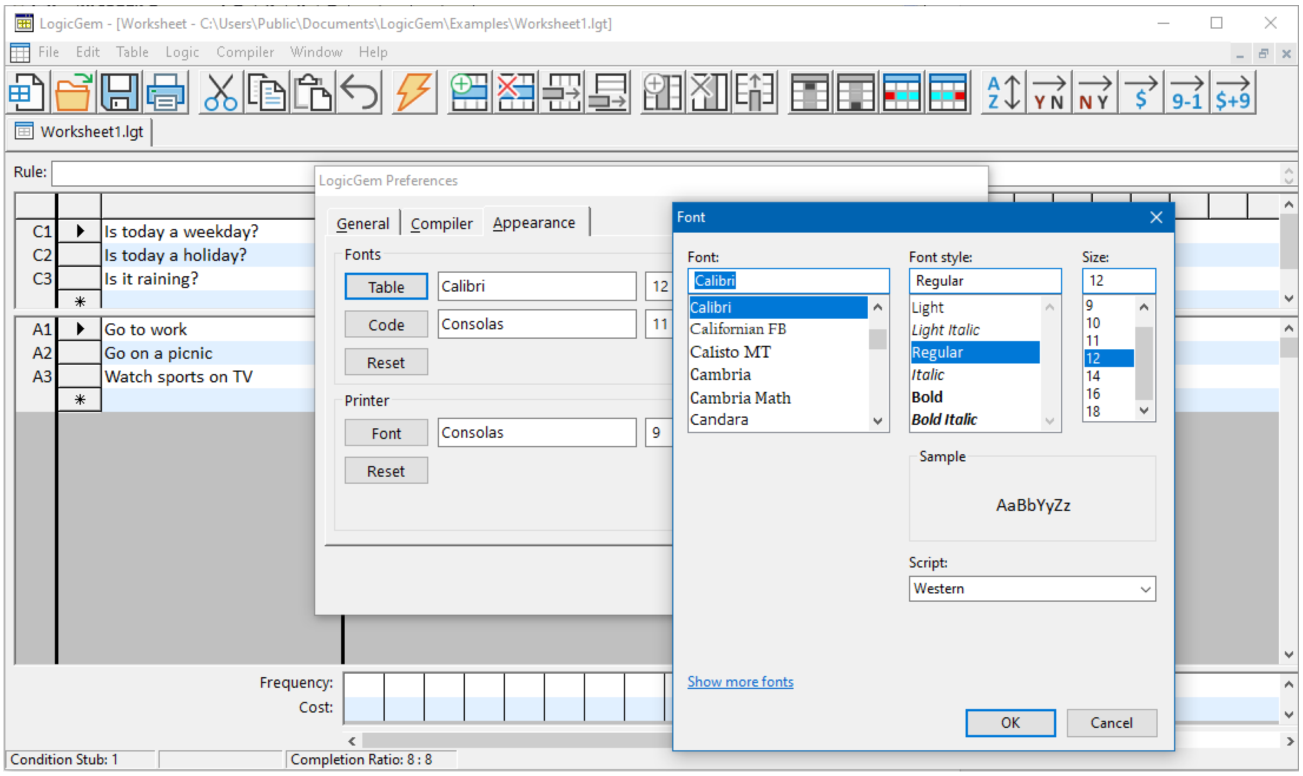
Colors

Color

Figure 41 – Color Palette for the Appearance Preferences Dialog Box

Font

The Font section of the Appearance preferences dialog box allows you to specify the font, font style, and size for displaying stub and entry values in the Worksheet. The default MS sans serif, regular, and 8-point settings may be changed by using the special font dialog as shown in the figure. By selecting the desired font attributes, and clicking the Apply button, the Worksheet will be displayed using the new font. It is recommended to use fonts 18 points in size or smaller. Above 18 points, text could be cut off in the display and other unusual problems could occur.



C1 Is today a weekday?
C2 Is today a holiday?
C3 Is it raining?

A1 Go to work
A2 Go on a picnic
A3 Watch sports on TV

Table: Calibri 12
Code: Consolas 11
Printer Font: Consolas 9

Font: Calibri (Calibri, Californian FB, Calisto MT, Cambria, Cambria Math, Candara)
Font style: Regular (Light, Light Italic, Regular, Italic, Bold, Bold Italic)
Size: 12 (9, 10, 11, 12, 14, 16, 18)
Script: Western

Figure 42 – Font Dialog of the Appearance Preferences Dialog Box

Exit LogicGem

How to exit

In order to end a LogicGem session you may use one of the following techniques:

Menu Bar

Click on the File menu function

Click on the Exit option.

Keyboard

Press Alt+F

Press X

Appendices

Appendix A - LogicGem Limits

When designing and implementing a new logic table, it is useful to know the limits upon which each table is bound. Table 9 itemizes the limits for the current version of LogicGem.

Description	**Limit Value**
Maximum number of condition stubs in a decision table.	15
Maximum number of characters in a condition stub.	1,024
Maximum number of action stubs in a decision table.	32,767
Maximum number of characters in an action stub.	1,024
Maximum number of characters in a rule description.	1,024
Maximum number of rules a decision table can reflect.	32,767

Table 9 – LogicGem Limits

Glossary

Action entry: a number horizontally adjacent to an action stub and vertically aligned with a column of one or more condition entries and possible other action entries.

Action label: leftmost column of the bottom half of a logic table. Action labels are normally A1-An in numeric sequence.

Action stub: the description field for target source language expressions (the action).

Ambiguous rules: two complex rules that can be decomposed into two overlapping sets of simple rules.

Ambiguous table: a logic table containing redundant rules and/or contradictory rules.

Column count: always either 1, if the current rules is simple (no dashes or NOT entries), or the number of simple rules implicit in a complex rule.

Compiler: See Logic Compiler.

Completion ratio: a ratio of distinct simple rules that *should* be in the logic table vs. the number that *are* in the table at any given instant.

Complex rule: a column of condition entries containing either one or more dashes (“don’t cares”), one or more capital letter entries (“NOT entries”) or a combination of the two.

Condition entry: a letter (y, n) or a dash (-) horizontally adjacent to a condition stub and optionally vertically aligned with a column of one or more other condition entries and/or action entries.

Condition label: leftmost column of the top half of a logic table. Condition labels are normally C1-Cn in numerical sequence.

Condition stub: the description field for target source language expressions (the conditions).

Condition value: See condition entry.

Contradictory rules: two or more rules in a logic table which invoke different actions for the same set of condition values. This makes it impossible to generate a correct program from the table.

Cost: this location indicator displays the current cost column.

Cost row: this optional line enables you to enter a sequence of numbers that suggest how costly a rule might be if it were invoked. These numbers enable the compiler to do optimizations relative to execution speed of the code structure produced. They are useful to the interpreter in logic tree construction.

Dashed entries: used in a logic table to indicate “don’t care” values. A “don’t care” in a rule means that any possible condition value can occur.

Decision table: See logic table.

Disambiguation: the removal of ambiguous rules.

Editor: See logic editor.

Entry: this cursor location indicator displays the current condition or action row and column.

Extended entry decision table: a table wherein condition entries may assume multiple values such as a, b, c, etc. (M>2). LogicGem does not currently support extended entry tables, a future version will.

Frequency: this cursor location indicator displays the current frequency column.

Frequency row: this optional line for a two digit number that estimates how often a rule might be expected to be invoked. These numbers enable the compiler to do optimizations relative to execution speed of the code structure produced. They are useful to the interpreter in logic tree construction.

Incomplete table: a logic table in which the number of simple rules is less than the product of the matrix values (for LogicGem, the matrix value is always 2).

Limited entry decision table: a table wherein condition entries may only contain two values, yes or no (M=2).

Logic compiler: compilers normally translate source code into machine code; LogicGem translate logic table into source code.

Logic editor: the portion of LogicGem that facilitates working with logic tables. The Logic Editor will generate a mechanically perfect set of simple rules for the user to edit then the Editor will eliminate the resulting superfluous columns.

Logic table: a logic table is an interactive graphic display, similar to a spreadsheet, representing the relations of combinations of conditions to combinations of actions.

Logically perfect table: a logic table in which the number of simple rules equivalent to it's complex rules is equal to the product of the matrix values (neither incomplete nor redundant).

Mechanically perfect table: a table which is logically perfect and contains no redundancies or contradictions.

Mixed entry decision table: a logic table wherein condition entries may assume either binary or multiple values (M>=2). LogicGem does not currently support mixed entry tables, a future version will.

NOT entries: a condition entry indicating that all values except the one specified are acceptable.

Reduce: a logic function that combines two rules into a single, more complex rule.

Redundant rules: two or more rules in a logic table which invoke the same actions for the same set of simple values. This means that they can be combined to generate a minimally correct table.

Rules: column of condition/action relations.

Rule name: a phrase that specifics in natural language what the rule is; useful for quickly grasping a column's meaning. Rule names can also become embedded comments in the code produced as output by the compiler.

Rule number: the column number at the too of a rule.

Simple rule: a rule without "don't cares" or NOT entries.