SYSTEM AND METHOD FOR GENERATING COMPUTER SOURCE CODE

Abstract: A system and method for generating computer source code using a document prepared by a non-programmer in a language and in a software application that the non-programmer is familiar with. The system and method parse out the program methodology prepared by the non-programmer and use it, in combination with pre-defined code snippets (30) to generate source code (26). The non-programmer, due to the real-time creation of the computer source code, is able to activate the program almost immediately after creation and dynamically test and modify the program as necessary (40).
before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments.

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.
SYSTEM AND METHOD FOR GENERATING COMPUTER SOURCE CODE

CROSS-REFERENCE TO RELATED APPLICATIONS

This application claims the benefit of U.S. Provisional Application No. 60/195,721, filed April 7, 2000, entitled "System And Method For Behaviorally Tailoring An Individual's Health-Related Behaviors", which is hereby incorporated by reference.

This application is related to U.S. Patent Application Serial No. 09/632,285, filed August 3, 2000, entitled "System And Method For Generating Tailored Media Content To Affect An Individual's Health-Related Behaviors", which is hereby incorporated by reference.

APPENDICES

The application includes computer program related materials in the form of appendices.

COPYRIGHT NOTICE

A portion of the disclosure of the patent document contains material which is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent document or the patent disclosure, as it appears in the U.S. Patent and Trademark Office patent files or records, but otherwise reserves all copyright rights whatsoever.

BACKGROUND OF THE INVENTION

Experts, either explicitly or implicitly, no matter what field of endeavor they specialize in, be it health, legal, financial, education, marketing, advertising or numerous others, apply some form of methodology or analysis to analyze a client's problem and determine the best way to address the problem. Much of the time the methodologies and analyses employed are complex, based on the experts' knowledge, training and years of experience and not readily adaptable as a computer program. However, in other instances, the expert applies a clearly defined sequential methodology or analysis to determine the cause of the problem and the best way to resolve it. In these instances, the methodology or analysis employed by the expert and the resulting advice is not much different than what a
computer does when it executes an algorithm. This fact has been known for years, and for years, people have tried to turn expert knowledge and methodologies into executable computer algorithms that provide solutions to a client's problems based on the data input by the client. This effort has enjoyed a substantial amount of success, but it has not been without difficulty and need for improvement.

Developing these expert methodology computer programs is difficult and time consuming. This stems mostly from the simple fact that the experts who develop the content and methodologies for the programs are often not computer programmers, and vice versa, the programmers are not experts in the field being addressed. Experts involved in such program development projects, historically, need to flowchart or describe the methodology and the generated output or solution in great detail. This flowchart or description is then provided to a computer programmer to generate source code that executes the flowcharted or described methodology. This system is workable, but it has its inherent limitations. After the programmer programs the algorithm, the code needs to be compiled and tested so the expert can review it to see if the program is working as intended. Often, however, the code is incorrect on the first try for a number of reasons. Many of these problems are caused by bugs that commonly occur in programming a new computer program. Many of these problems, more significantly though, are caused by the programmer misinterpreting the expert's flowchart or notes or the expert not explaining the methodology clearly or in enough detail. In these instances, the expert and the programmer need to meet, usually on numerous occasions, to re-write, re-run and review the code and its operation until it works properly. This is a very time consuming and inefficient, if not impossible, way to prepare computer programs.

As a result of these difficulties, there is a definite need for a system that allows a person to write computer code using language that the non-programmer is familiar with to create the code instead of having to learn complex computer program languages, such as C++ or Java. With such a system, subject matter experts are able to draft computer code directly, eliminating the inefficiencies and frustrations caused by using a middleman programmer to program the code. Some systems have been developed along these lines, but these systems still have numerous drawbacks and require significant training time. In one such system, the non-programmer creates a master content document (MCD) detailing the conditional parameters and their possible inputs and detailing the structure and content
of output that will be generated by a separate processing program. In such a system, the processing program, when executed, captures data from a system user and pulls in information from the MCD as necessary, based on these inputs. Such computer program generation systems, though easier and more efficient than prior processes, are still cumbersome and have numerous deficiencies and built-in inefficiencies. In the development of MCD program systems, a processing program still has to be generated by a computer programmer in addition to the MCD prepared by a non-programmer content expert. This processing program still needs to be reviewed and tested by both the programmer and the content expert to ensure that the MCD is compatible with the processing program and that the MCD and processing program are working properly together to generate the appropriate output. This system does not allow the non-programmer to view the results of his programming in real time. As a result, program development in an MCD system remains fairly labor and time intensive and is fairly inefficient.

MCD systems also have some very significant production limitations. Because of the structure of having the processing program operate on separately developed MCD, the condition statements that may be written into the processing programs for such MCD systems are fairly limited, and as a result relatively weak. Also, in such systems the output content that can be supported is fairly limited which ultimately restrains the final presentation of tailored output from the system. Also in production, as a result of a processing program operating on a separate MCD, the processing time for MCD systems is relatively slow, resulting in lengthy waits for tailored output.

Due to the shortcomings in prior systems, there is a need for a computer program generation system in which a non-programmer can write the methodology for a computer program in a language and in a software application that the non-programmer is familiar with, such as a word processing program or general text editor, and have this computer program methodology converted into an executable computer program. In conjunction with this need, there is a need for a computer program generation system in which the program developer can review the program in real time so he can ensure it is operating as intended and, if not, make the corrections immediately.

There is also a need for a computer source code generation system that can create programs having very powerful condition statements and having the ability to utilize a
wide variety of content (e.g., formatted text, pictures, tables, charts, graphics) from a variety of different sources. There is a further need for a computer source code generation system that can generate computer programs having the qualities discussed above that executes extremely quickly in a production environment.

SUMMARY OF THE INVENTION

According to one aspect of the present invention, a method for generating computer source code includes preparing a master tailoring document which contains logic for a computer program, inputting the master tailoring document into a document structure generator which parses out the computer program logic contained in the input document and builds a document tree structure based on the program logic, and providing a code writer program and a template of pre-defined source code snippets so the code writer program may retrieve the computer program logic from the document tree structure and combine the retrieved program logic with designated source code snippets from the template to generate source code for a computer program. The source code generation method may further include compiling the generated source code to convert the generated source code into executable code.

The source code generation method may also include providing computer memory and a computer workstation, moving the executable code to the computer memory, activating the executable code which generates a dynamic user interface on the computer workstation, inputting data into the dynamic user interface and submitting the data, processing the input data, and creating an output content document.

The source code generation method may further include formatting the output content document into a formatted file and transmitting the formatted file to the computer workstation for review. The act of formatting the output content document into a formatted file may be done by providing a hypertext markup language (HTML) publisher and an HTML layout file and by inputting the output content document and the HTML layout file into the HTML publisher to format the file into HTML format. The act of formatting the output content document into a formatted file may also be done by providing a print publisher and a print layout file and by inputting the output content document and the print layout file into the print publisher to format the file into a print file.
format. The act of formatting the output content document into a formatted file may also be done by formatting the output content document into a text file format.

In the source code generation method of the invention, the master tailoring document may also include commands that set document tree structure nodes when parsed by the document structure generator and that create branches from each generated node when parsed by the document structure generator. The master tailoring document may further include group commands which have subcommands that create branches from each generated node when parsed by the document structure generator.

The template of pre-defined source code snippets of the source code generation method may be written in extensible markup language (XML) and the generated source code may be written in the Java programming language. In another aspect of the present invention, the master tailoring document may include at least one condition statement written to output text content based on the condition statement. The master tailoring document may also include at least one condition statement written to output graphics content based on the condition statement.

According to another aspect of the present invention, a system for generating computer source code from a master tailoring document which contains logic for a computer program includes a code generator having a document structure generator, a memory, a code writer and a template of pre-defined source code snippets in which the document structure generator parses out the computer program logic contained in the master tailoring document and builds a document tree structure in memory based on the program logic and in which the code writer program retrieves the computer program logic from the document tree structure and combines the retrieved program logic with designated source code snippets from the template to generate source code for a computer program.

The source code generation system may further include a code compiler for compiling the generated source code to convert the generated source code into executable code. The master tailoring document of the source code generation system may include commands that set document tree structure nodes when parsed by the document structure generator and may also include commands that create branches from each generated node when parsed by the document structure generator. The master tailoring document of the source code generation system may also include group commands which have
subcommands that create branches from each generated node when parsed by the
document structure generator.

The template of pre-defined source code snippets of the source code generation
system may be written in extensible markup language (XML), and the generated source
code may be written in the Java programming language. According to another aspect of
the source code generation system, the master tailoring document may include at least one
condition statement written to output text content based on the condition statement. The
master tailoring document may include at least one condition statement written to output
drawings content based on the condition statement.

It is an object of the present invention to provide a computer source code
generating system and method in which a non-programmer can write the methodology for
a computer program in a language and in a software application that the non-programmer
is familiar with which generates source code for a computer program.

It is another object of the present invention to provide a computer source code
generation system and method in which the non-programmer may test, review and debug
the program he or she has developed in a dynamic fashion.

It is a further object of the present invention to provide a computer source code
generation system and method which produces computer source code capable of handling
powerful condition statements and capable of supporting a wide variety of content from a
wide variety of sources. It is another object of the present invention that the computer
source code generation system and method produce computer tailoring programs that
execute extremely quickly in a production environment.

BRIEF DESCRIPTION OF THE DRAWINGS

These and other features, aspects, and advantages of the present invention will
become better understood with regard to the following description, appended claims, and
accompanying drawings where:

FIG. 1 depicts an overview of the development process for generating computer
source code in the present invention;

FIG. 2 depicts an overview of the computer source code generated by the system
and method of the present invention in a production environment;

FIG. 3 illustrates an exemplary master tailoring document in an embodiment of the
present invention;
FIG. 4 illustrates the process within a code generator of the present invention for generating source code from the exemplary master tailoring document;

FIGS. 5A-M depict an exemplary code template used by a code generator of the present invention;

FIGS. 6A-G depict exemplary source code generated by the code generator of the present invention;

FIG. 7 depicts an exemplary dynamic user interface developed by executable code generated by the system and method of the present invention;

FIG. 8 depicts an exemplary output content document developed by executable code generated by the system and method of the present invention; and

FIG. 9 illustrates an exemplary display on the workstation of user of a system using the executable code generated by the system and method of the present invention.

DETAILED DESCRIPTION OF THE INVENTION

I. Overview

A. Program Development

Referring to Fig. 1, a content expert 20 interacts, via a computer workstation 22, with a code development computing system 24 to generate computer source code 26 and to test the code 26 to ensure it is working properly. The content expert 20 writes the algorithm content and logic into a text editing tool that he or she is familiar with, such text editing tools may be any text editor including word processing programs like Word or WordPerfect. The algorithm content and logic details the program’s needed inputs, the conditions and processes that the program applies and performs and the output information that the program generates after applying and performing these conditions and processes on the captured inputs. The entered program algorithm content and logic is saved as a document called a master tailoring document 28. The master tailoring document 28 maybe saved in a variety of formats, including Hypertext Markup Language (HTML) and Extensible Markup Language (XML). Saving the master tailoring document 28 in either HTML or XML format facilitates processing the information contained in the document.

Once developed, the master tailoring document 28 is input into the code development computing system 24. At that point, a code generator 30 operates on the master tailoring document 28, as described in detail below, to parse the master tailoring
document 28 into its component elements and then combine these component elements with templated source code to generate the outputted computer source code 26. The source code 26 generated by the code generator 30 is then compiled by a code compiler 32, resident in the code development computing system 24, which converts the source code 26 into computer executable code 34.

B. Dynamic Program Testing

Once compiled, the program can be tested in real time by the content expert 20. The content expert 20 does not have to wait for a programmer to write the code and compile the program before he or she can review it. This feature of the invention reduces program development time exponentially. Dynamic testing is possible because the developed executable code 34 is loaded into the memory 36 of the code development computing system 24 immediately after the executable code 34 is created. The stored program can then be accessed by the content expert 20.

To test the program, the content expert 20 activates the developed program executable code 34. The executable code 34, once activated, creates a dynamic user interface 40 on the content expert's workstation 22, as indicated by line 38. The dynamic user interface 40 prompts the content expert 20 for test input information. Once the content expert 20 enters all the necessary test information, he or she submits it to the program for processing, as indicated by line 42. The program, after receiving the inputs, processes the received information and, if the program executes properly, the program creates an output content document 44, as indicated by line 52. If, on the other hand, the program encounters an error during execution, it posts the error to an error log 54, as indicated by line 56, and generates an error message for the content expert's attention, as indicated by line 58. If the error was an improper data input, the content expert 20 can correct this easily and immediately by properly re-entering the input data and re-submitting it. If the error, however, was identified as a problem in the program code, the content expert 20 needs to review the master tailoring document 28 to determine the error and resolve it. In this situation, once the content expert 20 identifies the problem and corrects it, the content expert 20 needs to re-generate the computer source code 26 and re-compile it as described above.

The output content document 44 created by the program contains raw output data which may be formatted as required. In Fig. 1, the output content document 44 may be
formatted in three different ways. One way is through an HTML publisher 46. The HTML publisher 46 calls on an HTML layout file 48 to generate an HTML file 50 using the information from the output content document 44. The HTML layout file 48 is a previously created file developed by a web site developer in conjunction with the content expert 20 which defines the output format for the output content document information and defines and positions any graphics, charts or any other material required by the output content document to be inserted into the published document. As indicated by line 66, the content expert 20 can access the generated HTML file 50 through his workstation browser so he can review the output information and its presentation to determine if changes need to be made.

The output content document 44 may also be formatted as a text file 72 through a text publisher 70. The generated text file 72 may be transmitted to the content expert’s workstation 22, as indicated by line 74, where the content expert 20 may spell check the document and perform other text editing functions on it, if necessary. The output content document information may also be formatted as a print file 68, in a format such as PostScript. The print file 68 is generated by a print publisher 60 which calls on a print layout file 62 to generate the print file 68. The print layout file 62, similar to the HTML layout file 48, is a previously created file developed by a print layout artist in conjunction with the content expert 20 which defines the output format for the output content document information and defines and positions any graphics, charts or any other material required by the output content document to be inserted into the printed document. As indicated by line 64, the content expert 20 can access the generated print file 68 for review and correction, if necessary.

C. Program Use in a Production Environment

Referring to Fig. 2, once the program has been tested and certified for production, the developed program executable code 34 is moved to a program production system 100. Once in the production system 100, the program operates as it did in testing. As indicated by line 106, to access the program production system 100 and activate a system controller 107 resident in the program production system 100, a user (which may be any individual, including an end user or a system operator) uses a system access device 104. The system access device 104 may be any device capable of interacting with a system controller, such as a computer, a personal digital assistant (PDA), a cellular phone, a pager, a call center...
system, an interactive voice response ("TVR") system, a system for a data input center, an
e-mail system or any type of network access device. As line 108 indicates, the system
controller 107 creates a dynamic user interface which prompts the user for user input data.
Once the user enters all the necessary input data, he or she submits it to the system
controller 107 for processing, as indicated by line 110. The system controller 107, after
receiving the inputs, saves the inputs to a user input database 112. The system controller
107 then activates the developed executable code 34. The executable code 34 interacts
with the user input database 112, retrieving the pertinent information, and processes it.
The executable code 34 processes the input data to generate an output content document
114, as indicated by line 116.

As during testing, the output content document 114 created by the program
contains raw output data which may be formatted as required. In Fig. 2, the output content
document 114 may be formatted in three different ways. The HTML publisher 46 may
call on the HTML layout file 48 to generate an HTML file 118 using the output content
document information. The HTML file 118 is transferred to a web server 124 which may
interact with a browser on the user's system access device 104 which, as described above,
may be any number of devices including a computer, a PDA, a pager or a cell phone to
generate an end user web page 125 which displays the processed information for review
by the user.

The output content document 114 may also be formatted as a text file 120 through
a text publisher 70. The generated text file 120 may be transmitted to an e-mail exporter
126 for transference to the user's system access device 104 as an end user e-mail 127. The
output content document information may also be formatted as a print file 122, in such a
format such as PostScript. The print file 122 is generated by a print publisher 60 which
calls on a print layout file 62 to generate the print file 122. The print file 122 is transferred
to a File Transfer Protocol (FTP) server 128. The print file 122 may be sent to any
recipient 129 that needs the print file 122, such as a print vendor for printing. The print
vendor, in turn, may send the printed documents on to an end user 131.

II. Code Generation Details

The system of the present invention may be used to develop source code for any
type of programming language, such as Java, using a text editor program. In the preferred
embodiment, the content expert 20 develops a master tailoring document 28 similar to the
one depicted in Fig. 3 using a text editor program. The master tailoring document 28 is a set of tables that specify the program information. The tables specify the program name, the inputs that the program requires and the conditions that are applied to the input data to develop output content. To develop the program, the content expert 20 follows guidelines set for developing the program charts. An example of such guidelines are attached as Appendix A. To make program development simpler, the guidelines specify simple, intuitive, easy to follow commands and field identifiers.

Turning to the first table 150, the first column 152 of this table includes “command” cells. The command in the first cell of the first table is the “Document” command. The “Document” command requires the input of a document/program name in the adjoining cell in the second column. As indicated in cell 154, in this example, the program’s name is “SimpleTest”.

The next table in this example is the “Response” table 156. The Response table 156 specifies the input data that the program requires. The cells of the Response table may be formatted in any manner as long as they conform to a set of structuring guidelines similar to those attached as Appendix A. In this example, however, each row of the Response table is formatted in a similar manner. The first column 158 of the Response table 156, similar to the first column 152 of the first table 150, is a “command” column. The second column 160 of table 156 is a “name” column. When used with the “response” command, the “name” cell provides a name for the input that is requested by the “response” command. The third column 162 is an “options” column. A cell in the “options” column 162 lists any options required for that entry. For instance, if an input is required for a “response” field, the content expert 20 could ensure that the program would not continue unless data was entered for this field by specifying in the “options” cell that data had to be input for this field for the program to execute. The “options” cells of table 156 are not employed in the example depicted in Fig. 3. The fourth column 164 of table 156 is a “type” column 164. A cell in the “type” column 164 describes the characteristics a response to the query generated by the “Response” command must have. For instance, the “FirstName” variable 171 is a “String” type field as indicated by identifier 170 which means that only character values can be entered for this variable. The last column 166 of the table 156 is the “comment” column. A cell in the “comment” column 166 contains the prompt command that a user sees when he or she uses the program. For the Response
field 168, the user is prompted with the comment “First Name:” from cell 172. In the example depicted in Fig. 3, the program requests the user for his first name (field 168), his last name (field 174) and his age (field 176). The “FirstName” and “LastName” response fields both require character data input as indicated by the “String” designation in their respective “type” cells 170, 178. The “Age” response field 176 requires the input of an integer as indicated by the “int” designation in its “type” cell 180.

The next table 182 in this example is the “Content” table. The Content table 182 specifies the conditions and operations that are applied to the data inputs and the resulting outputs that are generated by these conditions and operations. As with the Response table, the cells of the Content table 182 may be formatted in any manner as long as they conform to a set of structuring guidelines similar to those attached as Appendix A. The first column 184 of the Content table 182, similar to the first columns 152, 158 of the first table 150 and the Response table 156, is a “command” column. The second column 186 of table 182 is a “name” column. The third column 188 is an “options” column. A cell in the “options” column 188, as with the Response table “options” cells 162, lists any options required for that entry. The “options” cells for table 182 are not employed in the example depicted in Fig. 3. The fourth column 190 of table 182 is a “condition” column 190. A cell in the “condition” column 190 specifies a boolean expression that, if true, generates a program specified output, such as text or graphics. The boolean expressions specified in a “condition” cell 190, in this embodiment, are written in standard Java language because these boolean expressions are inserted directly into the program source code that is ultimately generated. The fifth column 192 of table 182 has multiple uses depending on the type of command entered in the “command” column 184. As shown and explained in Appendix A, the content expert 20 may employ a variety of different commands in the “command” column 184 depending on what the content expert 20 wants to set a condition for. This format allows a content expert 20 to use very powerful conditions statements. For instance, if the content expert 20 uses the “Graphic” command described in Appendix A, frame 16, the generated program, when executed, will return a designated graphic if the condition statement is satisfied. Further, since Java code may be directly input into the master tailoring document 28 using the “Java” command (Appendix A, frame 20), the resulting program has all the power and capability of a program written directly in Java. This allows programs generated by the present invention to support a wide variety of
content, including formatted text, pictures, tables and charts. In the embodiment depicted in Fig. 3, “text” commands are illustrated so the entries in the fifth column 192 in this embodiment all specify text that the program outputs if the boolean expressions in the cells of the column condition 190 are true. Additional columns may be added to table 182 if necessary. Additional columns may include such things as alternate language versions (e.g., the text of column 192 translated into German, French or Spanish) which may be output instead of English if the user selects this alternate language as his or her primary language. (See the explanation of the “text” command, Appendix A, frame 16.)

In the example depicted in Fig. 3, the first programmed row 196 of “Content” table 182 defines a “Group”. The “Group” command defines the beginning of a series of commands that are to be grouped together. The “End” command defines the end of the grouped commands. The “Group” defined in the Fig. 3 example is the “Welcome” group as designated by the “name” cell 196. The second programmed row 200 of table 182 is a “text” command row. When this row is executed, the program generates the statement in cell 202, “Welcome $$FirstName$$”, where the first name entered by the program user (e.g., John) is output for the variable $$FirstName$$. The third programmed row 204 is also a “text” command row. When this row is executed, the boolean expression in cell 206 (Age<21) is evaluated. If the “Age” data input by the program user is less than 21, then the boolean expression is true and the program generates the statement in cell 208, “You are under 21.” The fourth programmed row 210 is a “text” command row with a boolean expression as well. When this line is executed, the boolean expression in cell 212 (Age>=21) is evaluated. If the “Age” data input by the program user is greater than or equal to 21, then the boolean expression is true and the program returns the statement in cell 214, “You are 21 or older.” The fifth programmed row 216 is an “End” command row which ends the “Group” designated in the “name” cell 218. In this example, the “End” command designates the end of the “Welcome” group.

Once the content expert 20 has completed the master tailoring document 28, the content expert 20 saves the master tailoring document 28. The master tailoring document 28 may be saved in a variety of formats including HTML or XML. Saving the master tailoring document 28 in either HTML or XML format facilitates further processing of the master tailoring document 28. Referring to Figs. 1 and 4, once saved, the master tailoring document 28 is input into the code generator 30. A document structure generator 250
residential in the code generator 30 operates on the input master tailoring document 28. The document structure generator 250 parses the information contained in the master tailoring document 28 and builds a tree structure representing the master tailoring document structure in memory 252. The document structure generator 250 creates the tree structure based on the commands in the input master tailoring document 28. When a “Document” command is encountered, the document structure generator 250 sets the beginning or “root” of a document tree. As depicted in Fig. 4, for the master tailoring document 28 from the Fig. 3 example, the “Document/Simple Test” node object 254 is generated from the master tailoring document “Document” command 152. The node 254 sets the root for the SimpleTest document tree structure. For each master tailoring document command encountered by the document structure generator 250 subsequent to the “Document” command, the document structure generator 250 generates branches and sub-branches from the “Document” node 254 until the end of the master tailoring document 28 is reached.

As the document structure generator 250 operates on the master tailoring document 28 of Fig. 3, it builds nodes for each “response” and “group” command encountered. Certain commands, such as “response”, “variable” and “parameter”, generate independent nodes off of the “document” node. Other commands, such as “group” and “level”, generate root nodes for a sub-branch from the “document” node, and other commands, such as “text”, “graphic” and “table”, generate object nodes which have the characteristics specified in the master tailoring document 28. In the example depicted in Fig. 4, the “document” node “SimpleTest” 254 has three “response” nodes 256-260 and one “group” node 262 extending from it. The “response” nodes 256-260 have the characteristics specified in the master tailoring document 28. The “group” node 262 sets the node for the “Welcome” group of commands. Extending from the Welcome “group” node 262 are three “text” nodes 264-268 and one group “end” node 270. The “text” nodes 264-268 have the characteristics specified by the master tailoring document 28 depicted in Fig. 3.

Once the tree structure is developed in memory 252, the code generator 30 generates the source code 26. The code generator 30 generates the source code 26 by invoking a code writer program 272. A sample code writer program 272 is attached as Appendix B. The code writer program 272 generates source code 26 by selectively combining the objects of the generated master tailoring document tree structure with pre-
defined, standardized source code snippets from a template 274 to generate a customized program.

Referring to Figs. 4-6, the process by which the code writer 272 generates source code 26 is illustrated. The code writer 272 first generates a common preamble 300 in the generated source code 26 of Fig. 6A. The common preamble 300 is generated by selecting a series of preamble snippets 302 from the code template 274 of Fig. 5A. In the preferred embodiment, the code template snippets are formatted in XML. XML is preferable since the XML format is a pure data format that is more amenable to manipulation than languages that combine data with representation characteristics such as HTML. The code writer 272 operates on the pre-defined preamble snippets 302 to insert the proper objects from the master tailoring document tree structure at the appropriate points. For instance, referring to Figs. 5A and 6A, the code writer 272 inserts the name of the “document” node 254 (“SimpleTest”) into the variable “{0}” 304 in the code snippet 302 to create a Java class named “SimpleTest” 306. After creating the preamble, the code writer 272 next writes any Java language that was included directly in the master tailoring document 28, by way of the “Include” command, into the source code. The code writer 272 uses the “include” code snippets 304 (Fig. 5B) from the code template 274 to create this source code, as block 308 indicates (Fig. 6A). The “Include” command was not used in the example depicted in Figs. 3-6, but is explained in the master tailoring document guidelines attached as Appendix A. After inserting any “included” Java language into the generated source code 26, the code writer 272 writes any global Java language that was included directly in the master tailoring document 28, by way of the “Java” or “Function” commands, into the source code. The code writer 272 uses the “java” or “function” code snippets 306 (Fig. 5B) from the code template 274 to create this source code, as block 310 indicates (Fig. 6A). Neither the “Java” nor the “Function” commands were used in the example depicted in Figs. 3-6, but these commands are explained in the master tailoring document guidelines attached as Appendix A.

After the code writer 272 writes any “Java” or “Function” language that was written in the master tailoring document 28, the code writer 272 writes a series of Java methods using the document tree structure and the code template snippets. As block 312 (Fig. 5C) of the code snippet template 274 and block 314 (Fig. 6A-B) of the generated source code 26 indicate, the first method written is named “getResponseNames” and it sets
the names of the "response" variables in the generated code by retrieving the "response" variable names from the document tree structure and inserting them into the code template snippet 312 at variable "{1}" designated as 316. As illustrated in Fig. 6B, the "response" variables in the Java code of this example are "FirstName" 318, "LastName" 320 and "Age" 322 which come from the three "response" nodes 256-260 of the document tree structure depicted in Fig. 4.

As block 324 of the code snippet template 274 (Fig. 5D) and block 326 of the generated source code 26 (Fig. 6B) indicate, the next method written is named "getInstrumentNames" and it sets the names of the "instrument" type variables in the generated code by retrieving any "response", "label" or "endlabel" variable names from the document tree structure and inserting them into the code template snippet 324 at variable "{1}" designated as 328 in the code snippet 324. As illustrated in Fig. 6B, the "instrument" variables in the Java code of this example are "FirstName" 330, "LastName" 332 and "Age" 334 which come from the three "response" nodes 256-260 of the document tree structure depicted in Fig. 4. There are no "label" or "endlabel" variables used in the example depicted herein.

As block 336 of the code snippet template 274 (Fig. 5D-E) and block 338 of the generated source code 26 (Fig. 6B) indicate, the next method written is named "validateResponseValue" and it validates that the variable "name" equals the name of the retrieved "response" variables. This code is generated by retrieving the "response" variable names from the document tree structure and inserting them into the code template snippet 336 at variable "{0}" as indicated at 340. Again, the "response" variables in the Java code of this example are "FirstName" 342, "LastName" 344 and "Age" 346.

As block 348 of the code snippet template 274 (Fig. 5E) and block 350 of the generated source code 26 (Fig. 6B-C) indicate, the next method written is named "getOption". This method retrieves and sets any "option" preferences that were designated for a specified "response" variable in the master tailoring document 28 in cell 162 (e.g., if the content expert 20 in the master tailoring document 28 designated that a "response" variable required an input from the system user, this would be retrieved by the "getOption" method at this point.) This code is generated by retrieving the "response" variable names from the document tree structure and inserting them into the code template snippet 348 at variable "{0}", indicated at 351. The code writer 272 also retrieves the
“option” characteristics for each “response” variable node in the document tree structure and inserts these “option” characteristics into the code template snippet 348 at variable “{1}”, indicated at 352. As illustrated in Fig. 6C, the “response” variables of the example explained herein do not have “option” characteristics so no “options” are set for the “response” variables “FirstName”, “LastName” and “Age” as indicated by designators 354-358.

As block 360 of the code snippet template 274 (Fig. 5G) and block 362 of the generated source code 26 (Fig. 6C) indicate, the next method written is named “checkRequired” and it checks, based on the “option” information retrieved above, whether a designated “response” variable requires a response. This code is generated by retrieving the “response” variable names from the document tree structure and inserting them into the code snippet template 360 at variable “{0}” as indicated at 364. The code writer 272 also sets the boolean expressions, for this example, to false at variable “{1}” as designated by 366 as indicated at 368-372 of the Java source code (Fig. 6C) because an input was not required by the content expert 20 for each of the “response” variables.

As block 374 of the code snippet template 274 (Fig. 5G-H) and block 376 of the generated source code 26 (Fig. 6C) indicate, the next method written is named “getVariableNames”. The “getVariableNames” method sets the names of the “variables” designated by the content expert 20 in the generated code by retrieving the “variable” names from the document tree structure and inserting them into the code template snippet 374 at variable “{1}” at 378. The example explained herein does not have any “variables” designated by the content expert 20 so there are no “variables” incorporated into the Java source code as the blank space at 380 indicates.

As block 382 of the code snippet template 274 (Fig. 5H) and block 384 of the generated source code 26 (Fig. 6C) indicate, the next method written is named “getParameterNames”. The “getParameterNames” method is very similar to the “getVariableNames” method. The “getParameterNames” method sets the names of the “parameters” designated by the content expert 20 in the generated code by retrieving the “parameter” names from the document tree structure and inserting them into the code template snippet 382 at variable “{1}” at 386. The example explained herein does not have any “parameters” designated by the content expert 20 so there are no “parameters” incorporated into the Java source code as the blank space at 388 indicates.
As block 390 of the code snippet template 274 (Fig. 5H) and block 392 of the generated source code 26 (Fig. 6C-D) indicate, the next method written is named “computeVariable”. The “computeVariable” method calculates the value of any “variables” or “parameters” defined in the master tailoring document 28 by the expert 20. The code for this method is generated by retrieving the “variable” names from the document tree structure and inserting them into the code template snippet 390 at variable “{0}” indicated by 394 (Fig. 5I). Any “variable” computational characteristics are retrieved from the associated “variable” nodes in the document tree structure and inserted into the code template snippet 390 at variable “{1}” indicated by 396. Since the example explained herein does not have any “variables” or “parameters” designated by the content expert 20, there are no “variables” or “parameters” to compute as the blank space in the Java source code at 398 indicates (Fig. 6D).

As block 400 of the code snippet template 274 (Fig. 5I) and block 402 of the generated source code 26 (Fig. 6D) indicate, the next method written is named “setVariableValue”. The “setVariableValue” method sets the value of a given variable based on a user input. The code for this method is generated by retrieving the “response” variable names from the document tree structure and inserting them into the code template snippet 400 at variable “{0}” indicated by 404. The result in the source code is that the “response” variables are set to equal the user’s inputs for each “response” variable. For instance, the “FirstName” variable 406 equals the “value” 408 input by the user for the First Name; the “LastName” variable 410 equals the “value” 412 input by the user for the Last Name; and the “Age” variable 414 equals the “value” 416 input by the user for his or her age.

The next series of code snippet templates 418-426 and the related generated source code 428-436 define “get” methods that are generated in the same manner. The code writer 272 generates five “get” methods: 1) “getVariableValue”, 2) “getVariableType”, 3) “getCommand”, 4) “getLabel” and 5) “getObject”. The “get” methods are generated by retrieving the “variable” names from the document tree structure and inserting them into the code template snippets 418-426 (5J-K, 5E-G) at variable “{0}” and then defining what the program is to provide in the “return” line at variable “{1}”.

The output from the program is generated by a “write” method. As block 438 of the code snippet template 274 (Fig. 5K) and block 440 of the generated source code 26
(Fig. 6F) indicate, the “write” method is named “writeDocument” and it calls on lower level methods (e.g., “writeSimpleTest” 442) to produce the program output. The “writeDocument” method is generated by retrieving the “document” name from the document tree structure and inserting it into the code template snippet 438 at variable “{0}” indicated by 444. The lower level methods are generated by retrieving “group” objects branching from the “document” node and inserting them in the proper type of code snippet in the code template. Snippet 438a formats a text node. Snippet 438b formats a graphics node. Snippet 438c formats a group node. Snippet 438d formats a content node. Snippet 438e formats a table node. Snippet 438f formats a chart node. Snippet 438g depicts a series of formats for selection nodes. Snippet 438h depicts a group method format snippet.

In the example program, the code writer 272 develops a group method named “SimpleTestWelcome”. The first step in generating group method “SimpleTestWelcome” is accomplished by retrieving the name of the “group” being operated on from the document tree structure (Fig. 4) and inserting the “group” name into the variable “{0}” as indicated at 446 (Fig. 5L), 448 (Fig. 6F). The code writer 272 then finds the next node of the group “Welcome” that needs to be inserted into the group method. In this example, it is the text node 264 (“Welcome $\$FirstName$$\$). The code writer 272 then inserts the text from the text node into the text node formatting snippet 438a (Fig. 5K) to generate a text output line of code as indicated at 450 (Fig. 6F). The code writer 272 then operates on the next node of the group, node 266, a conditional text node (“If Age<21, return: “You are under 21.’”). The code writer 272 inserts this conditional statement into the appropriate selection node formatting snippet 438g (Fig. 5L) to generate the conditional code found at 452. The code writer 272 then operates on the next node of the group, node 268, another conditional text node (“If Age>=21, return: “You are 21 or older.”) The code writer 272 inserts this conditional statement into the appropriate selection node formatting snippet 438g (Fig. 5L) to generate the conditional code found at 454. The code writer 272 then encounters the group “end” node 270 for the “welcome” group which triggers the code writer 272 to finish processing and preparing the SimpleTestWelcome group code. The code writer 272 continues on to write a few more administrative methods that are generated in a manner similar to the ones described above.
At this point the code writer 272 has generated a complete Java source code document. Referring again to Fig. 1, the code development computing system 24 then compiles the generated source code 26 in the code compiler 32. The code compiler 32 converts the generated source code 26 into executable code 34. The executable code 34 is moved to memory 36 where the compiled executable code 34 can be tested by the content expert 20. The content expert 20 may test the generated program by requesting the code development program computing system 24 to invoke the compiled executable code 34. The activated executable code 34 develops the dynamic user interface 500 as depicted in Fig. 7. The dynamic user interface 500 is generated by the code development computing system 24 by calling the “get” methods of the generated source code 26 (the “getVariableValue” method 428, the “getVariableType” method 430, the “getCommand” method 432, the “getLabel” method 434, the “getObject” method 436). The dynamic user interface 500 appears on the content expert’s workstation 22 and identifies field information for the content expert 20 to fill in. In the source code generated in this example, the dynamic user interface 500 requests the content expert 20 to input a first name in field 502, a last name in field 504 and an age in field 506. In this example, the content expert 20 inputs a first name of “John”, a last name of “Doe” and an age of “22”. Alternatively, this input data may be called from a stored data file and input into the generated program. The input data is sent from the content expert’s workstation 22 to the program executable code 34 for processing. The first thing the program does is validate that the proper type of data was entered for each data field (e.g., only numbers, not characters, were entered for an integer “int” type field). If the input data is not valid, the code development computing system 24 sends an error message back to the content expert 20. The content expert 20 then corrects the input data and re-submits it. Once all of the data submitted is valid, the executable code 34 of the generated program passes all of the input data into its respective Java variables (e.g., the first name information (John) is stored in the “FirstName” variable). Once all of the inputs are loaded, the generated program calls the “writeDocument” method which processes the input data and prepares an output content document 44. In the preferred embodiment, the output content document 44 is an XML document which facilitates further processing. The “writeDocument” method first calls the “writeDocumentHeader” method 460 which writes a document header 462 for the output content document 44 depicted in Fig. 8. The
"writeDocument" method next calls the "writeSimpleTestWelcome" method at designation 442. The "writeSimpleTestWelcome" method 456 first calls method "writeGroupHeader" 448 which writes the group header 464 to the output content document 44. The "writeSimpleTestWelcome" method 456 then calls the "writeText" method 450 which retrieves the input for the "FirstName" variable, which in this case is "John" and writes the formatted text section 466 of the output content document 44 which contains the statement "Welcome John". The "writeSimpleTestWelcome" method 456 next calls the selection method 452 which retrieves the input for the "Age" variable, which in this case is "22", and determines whether the input age is less than 21. In this case, it is not, so the "writeSimpleTestWelcome" method 456 skips to the next selection method 454 which determines if the input for the "Age" variable is greater than or equal to 21. In this case the "Age" variable is greater than 21, so the selection method 454 writes the formatted text section 468 of the output content document 44 which contains the statement "You are 21 or older." The "writeGroupTrailer" method 470 then writes the trailer 472 for the "group" element in the output content document 44. Finally, the "writeVariableValues" method 474 writes and formats all of the variable information to the output content document 44.

The output content document 44, since it is saved in a pure data format such as XML, may be formatted into any presentation format that the system operator wants to put the output information into. Referring again to Fig. 1, the manner in which the output information in this example may be formatted is depicted. The output information may be saved as a HTML file 50 by conversion through the HTML publisher 46. The HTML file 50 is then sent to content expert’s workstation 22 where the workstation’s browser opens the HTML file 50 and displays it for the content expert 20 to review. Fig. 9 illustrates what the content expert 20 sees in the example described herein. In a similar manner, the output content document 44 may be sent through the text publisher 70 to generate a text file 72 for the content expert 20 to review or through the print publisher to generate a print file 68 for the content expert 20 to review.

While the invention has been discussed in terms of preferred and specific embodiments, it should be appreciated by those of skill in the art that the invention is not so limited. The embodiments are explained herein by way of example, and there are
numerous modifications, variations and other embodiments that may be employed that would still be within the scope of the present invention.
Appendix A
Master Tailoring Document Format
10/28/99
# Table of Contents

Introduction .............................................................................................................. 3
MTD Format .................................................................................................................. 3
   MTD Files .................................................................................................................. 3
   Command Cell ........................................................................................................... 3
   Name Cells ............................................................................................................... 4
   Options Cells ............................................................................................................ 4
   Java Type Cells ....................................................................................................... 4
   Java Code Cells ....................................................................................................... 5
   Condition Cells ....................................................................................................... 5
   Text Cells ............................................................................................................... 5
   Label Cells .............................................................................................................. 6
Feature Descriptions .................................................................................................. 6
   Java Substitution ...................................................................................................... 6
   Math Support ........................................................................................................... 6
   Date Variables ......................................................................................................... 7
   TList Variables ....................................................................................................... 7
   Embedded Styles .................................................................................................... 8
   Min/Max Options ..................................................................................................... 9
   Tables and Charts ................................................................................................. 9
   Bulleted Lists ......................................................................................................... 9
Command Definitions .................................................................................................. 9
   DOCUMENT Command .......................................................................................... 9
   RESPONSE Command ......................................................................................... 10
   PARAMETER Command ....................................................................................... 10
   VARIABLE Command ........................................................................................... 11
   GROUP Command .................................................................................................. 11
   END Command ...................................................................................................... 11
   LEVEL Command .................................................................................................. 12
   TEXT Command .................................................................................................... 13
   GRAPHIC Command ............................................................................................ 13
   CONTENT Command ............................................................................................ 13
   SECTION Command .............................................................................................. 14
   TOPIC Command ................................................................................................. 14
   TABLE Command .................................................................................................. 14
   CHART Command .................................................................................................. 15
   SENTENCE Command ........................................................................................... 16
   FUNCTION Command ............................................................................................. 17
   JAVA Command ....................................................................................................... 17
   INCLUDE Command ............................................................................................... 17
   SCHEDULE Command ............................................................................................ 18
   COMMENT Command ............................................................................................. 18
   COMMAND and VERSION Commands ................................................................... 18
   LABEL Command ................................................................................................... 18
Built-In Functions ...................................................................................................... 20
Special Characters ...................................................................................................... 24
Prototype MTD File .................................................................................................... 27
   Document & Version ............................................................................................ 27
   Java ......................................................................................................................... 27
   Responses ............................................................................................................... 27
   Variables and Parameters ..................................................................................... 28
   Introduction Section .............................................................................................. 28

25
Introduction

This document describes the format of a "Master Tailoring Document" or "MTD". The MTD format supports the new Tailored Java (TJ) approach to tailored messaging. An MTD file consolidates many tailoring-related items into a single file, thus eliminating many potential areas of confusion and miscommunication.

MTD files are processed by the new "TJBuilder" application. In fact, TJBuilder is incorporated into the Grinder tool so that content developers may never actually need to run it directly.

MTD Format

MTD Files

MTD files are created using Microsoft Word. Within the MTD file, "tables" are used to contain tailoring information. TJBuilder ignores anything outside of a table. It is a good practice to place commentary text in the MTD file but outside of any table. Any number of tables may be used in the MTD file. There is no significance to ending one table and starting another. The format of the tables is not significant. For example, developers may select any column width that is appropriate.

The MTD files are saved in HTML format for processing in TJBuilder/Grinder. Because MS Word 97 files saved as HTML do not retain all document information, MTDs will be constructed and maintained as "doc" files and then converted to HTML for processing. Once HMI begins using Word 2000, in which documents saved in HTML format retain all Word information, MTD files will be created and maintained in HTML.

TJBuilder looks for a "command word" in the first cell of each row. This command word determines the format of the rest of the cells in that row.

Command Cell

The first cell in a row within a table contains the "Command Word." For some commands, additional information may be included after the command word (e.g. "LEVEL 1"). All currently defined commands are further described starting on page 9. Below is a summary:

- DOCUMENT specifies the document name and title.
- RESPONSE defines a questionnaire response.
- PARAMETER defines a characterization variable based on RESPONSE variables and on other PARAMETER variables. PARAMETER variables are loaded from and saved into the database.
- VARIABLE defines a variable just like PARAMETER. However, VARIABLE variables are not saved into the database.
- GROUP collects the commands that follow it into a group.
- END terminates a GROUP.
- LEVEL groups a collection of commands together at a specified level.
- TEXT specifies content text.
- GRAPHIC specifies a graphic.
- TABLE creates data to be displayed in tabular format.
- CHART creates data to be displayed as a chart.
- CONTENT synthesizes either text or graphics using Java code.
- SECTION is equivalent to "LEVEL 1".
- TOPIC is equivalent to "LEVEL 2".
- SENTENCE is equivalent to TEXT.
- FUNCTION defines a Java function that may be referenced in subsequent code.
- JAVA allows raw Java code to be included. This code might define additional variables or Java functions. This is an advanced feature that will often not be used.
- INCLUDE specifies an additional file to include at the current table position.
- SCHEDULE allows the use of Java code for determining when a document for a particular enrollee should be tailored.
- COMMENT defines a comment that will be ignored.
- COMMAND and VERSION are similar to COMMENT. COMMAND may be used to place headers at the top of a table to label columns for ease of editing. VERSION may be used to record modifications.
- LABEL defines a label used for formatting a user interface (used by Grinder).

Commands may be in any combination of upper and lower case. If a command is not recognized, a warning message is generated.

Any command may be prefixed by "/", e.g. "//INCLUDE". This will not generate an error but will disable that command, essentially treating it like a comment. This is useful for temporarily disabling a command without otherwise disturbing it.

**Name Cells**

Many commands, such as RESPONSE and TEXT, use the second column to contain a name. For content commands like TEXT, you can leave the name cell blank and TjBuilder will create a name.

The text in the name cell must be a single "word" in that it can have no spaces. It also must not have a paragraph end (\‘\‘) within the cell. (You can see these by using the Show/Hide \‘\‘ command.) The name must also start with a letter. Upper/lower case is significant.

Finally, you cannot reuse the names. The TjBuilder program will detect duplicate names and generate an error message.

**Options Cells**

Many commands use the third column to contain a list of options. This is generally a list of keywords separated by commas. Some options may also have a value. Example:

```
required, width=10
```

See the individual commands for more information on options.

**Java Type Cells**

All the variable definition commands (RESPONSE, PARAMETER, VARIABLE) take a Java type in the fourth column. The supported types are:

<table>
<thead>
<tr>
<th>Type/Format</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>true or false</td>
</tr>
<tr>
<td>byte</td>
<td>integer values from 0 to 255</td>
</tr>
<tr>
<td>char</td>
<td>Unicode characters (e.g. &quot;A&quot;)</td>
</tr>
<tr>
<td>short</td>
<td>integer values from -32768 to 32767</td>
</tr>
<tr>
<td>int</td>
<td>integer values from -2147483648 to 2147483647</td>
</tr>
<tr>
<td>long</td>
<td>integer values from $-2^{63}$ to $2^{63}$</td>
</tr>
<tr>
<td>float</td>
<td>32 bit floating point</td>
</tr>
<tr>
<td>double</td>
<td>64 bit floating point</td>
</tr>
<tr>
<td>String</td>
<td>strings of char values</td>
</tr>
<tr>
<td>Date</td>
<td>date and time</td>
</tr>
<tr>
<td>TList min,max,a,b,c,d</td>
<td>list of items, valid with it has at least min items but not more that max. See below.</td>
</tr>
</tbody>
</table>
**Java Code Cells**

Some cells contain "snippets" of Java code, i.e. conditions, characterization, etc. This Java code will be inserted into the created Java source file and will then be compiled by the Java compiler. As a result, the Java snippets must legal Java code with one exception.

MS Word allows the use of "smart quotes." As the user types a " character, a " or " is inserted instead, depending on the context. This is very useful for writing content text. Often it is also necessary to use a " in Java code but the use of " and " is not legal. In order to allow the content developer to use smart quotes when writing content and in order to avoid turning the smart quote option on and off, it will be allowable to use " and " in Java code. TJBuilder will automatically replace them with " . Many of the examples in this document use " and " in Java code.

**Condition Cells**

A condition cell contains Java code that controls whether content will be included in the tailored document. If the condition cell is blank, the text will always be included. Otherwise, the condition must be written as a standard Java "expression" that evaluates to a boolean value. For example, a boolean variable by itself is a valid boolean expression. Any comparison (e.g., Age>=21) is also a valid boolean expression.

The following is a brief introduction to the types of boolean expressions possible in conditions. More details on writing Java code are outside the scope of this document.

<table>
<thead>
<tr>
<th>Simple cases:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Age &gt;= 21</td>
<td>The text/graphic will be included if the value of the Age variable is greater or equal to 21.</td>
</tr>
<tr>
<td>Male=true</td>
<td>Included if the boolean variable Male is true.</td>
</tr>
<tr>
<td>Male=0</td>
<td>Same thing since Male is a boolean itself.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Compound cases:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>DiseaseCount==1 &amp; Gender!=null</td>
<td>Included if the value of DiseaseCount is greater than one and the Gender is known.</td>
</tr>
<tr>
<td>(Age==0</td>
<td></td>
</tr>
</tbody>
</table>

**Text Cells**

The TEXT and SENTENCE commands contain a text cell. Text in this cell may be included in the TCD, depending on the expression in the condition cell. Within the text cell, you may use bold, italic and underline. This formatting is carried through to the final output. Other changes in format may be specified using the embedded style feature described below.

Here are the rules for the contents of text cells:

- You may include paragraph ends (¶) in the text. These will be carried through to the final output.

NOTE: A bug in MS Word causes a ¶ that occurs at the very beginning of a cell to be lost when saving to HTML. Placing a space character in front of the ¶ prevents this problem. In other words:

<table>
<thead>
<tr>
<th>TEXT</th>
<th>¶</th>
<th>This booklet was designed using computer technology</th>
<th>This does not work, the ¶ is lost.</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEXT</td>
<td>¶</td>
<td>This booklet was designed using computer technology</td>
<td>This does work, because there is a space before the ¶.</td>
</tr>
</tbody>
</table>

This problem is corrected in Word 2000.

- You may use straight double-quotes (") and single quotes (') anywhere within the text.
- MS Word 97 converts directional double-quotes (" and ") into straight double-quotes when saving to HTML. In order to use directional double-quotes, insert #i#dq## and #r#dq## in the text for left and right double-quotes. Word 2000 does not have this problem.
- Double dollar characters ($$) may not be used unless it is for Java substitution as defined in the next Java Substitution section on page 6.
- Double pound characters (##) may not be used unless it is to pass information to the publication engine, e.g. ###lq### for left quotation or ###style for embedded styles.
• Other special characters (e.g. ©) may be used as long as they are in the special characters table on page 24.
• Bulleted lists (see below).

Label Cells

The LABEL and RESPONSE commands allow the placing of a "label" in column 6. This label is used by the Grinder development tool to label the input user interface. When used on a LABEL command, Grinder displays the label text on a line by itself in the user interface. Grinder also provides a popup menu of all LABEL commands to assist in navigating through the user interface.

When use in a RESPONSE, the label text is displayed next to the input field in the Grinder user interface.

Feature Descriptions

Java Substitution

Variable text may be substituted into output by including Java code right in the text by enclosing it in $$\text{Java}$$$. The enclosed Java code must evaluate to either a String, boolean or number. For example:

<table>
<thead>
<tr>
<th>TEXT</th>
<th>Your age is SS$\text{Age}$$.</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEXT</td>
<td>Your birthday is SS$\text{dateToFullString(BirthDate)}$$.</td>
</tr>
<tr>
<td>TEXT</td>
<td>It has been SS$\text{dateToDay(today)}$$.SS$\text{dayToDay(Qui}$$.SS$\text{tDate)}$$.SS$\text{ days since you quit smoking.}</td>
</tr>
</tbody>
</table>

In the first example, the value in Age will be substituted for "SS$\text{Age}$$" resulting in something like:

Your age is 39.

In the second example, "dateToFullString(BirthDate)" is Java code that converts the Date variable named "BirthDate" to a String. The following might result:

Your birthday is Tuesday, April 12, 1952.

In the third example, the "today()" function is called and it returns a Date containing the current date. It is converted to a day number by "dateToDay()". Similarly, the QuitDate variable is converted to a day number and the difference is computed by the subtraction sign "-". The resulting numeric value is automatically converted to a String and inserted into the tailored text. The following might result:

It has been 14 days since you quit smoking.

See the section below for more information on Date variables.

Math Support

The CONTENT command can be used to do simple or complex math. Consider the following example:

<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>TotalCalories</th>
<th>int</th>
<th>TotalCalories = 0;</th>
</tr>
</thead>
<tbody>
<tr>
<td>GROUP</td>
<td>Recipe1</td>
<td>IncludeRecipe1</td>
<td></td>
</tr>
<tr>
<td>TEXT</td>
<td></td>
<td>Recipe for apple dumplings...</td>
<td></td>
</tr>
<tr>
<td>CONTENT</td>
<td></td>
<td>TotalCalories += 850;</td>
<td></td>
</tr>
<tr>
<td>END</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP</td>
<td>Recipe2</td>
<td>IncludeRecipe2</td>
<td></td>
</tr>
<tr>
<td>TEXT</td>
<td></td>
<td>Recipe for banana bread...</td>
<td></td>
</tr>
<tr>
<td>CONTENT</td>
<td></td>
<td>TotalCalories += 920;</td>
<td></td>
</tr>
<tr>
<td>END</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP</td>
<td>Recipe3</td>
<td>IncludeRecipe3</td>
<td></td>
</tr>
<tr>
<td>TEXT</td>
<td></td>
<td>Recipe for chocolate cake...</td>
<td></td>
</tr>
<tr>
<td>CONTENT</td>
<td></td>
<td>TotalCalories += 1100;</td>
<td></td>
</tr>
</tbody>
</table>
In the above, the VARIABLE command defines a variable named TotalCalories to contain the number of calories. Any combination of the three groups can be selected, depending on the values of IncludeRecipe1, IncludeRecipe2 and IncludeRecipe3. In any case, TotalCalories will end up containing the total number of calories.

**Date Variables**

In addition to "int", "boolean", etc, the "Date" type can be used to store a date. This date might be imported from a questionnaire or may be computed. Often it is very useful to convert a Date to a day number using the "dateToDay()" function. This function returns an "int" value that is the number of days since January 1, 1970. As it may be negative, dates before January 1, 1970 are handled properly. For example, to compute the number of days since a person has quit smoking:

```
VARIABLE DaysSince int
DaysSince = dateToDay(today()) - dateToDay(QuitDate);
```

As you might guess, "today()" returns a Date with the current date. To compute a date, you can use "dayToDate":

```
VARIABLE PSG1Date Date
PSG1Date = dayToDate(dateToDay(QuitDate)+14);
```

Here QuitDate is converted to a day number, 14 is added to it and then that value is converted back to a Date.

**TList Variables**

A TList variable contains a list of items, each item described by a text string. The TList variable also contains minimum and maximum limits. While it is possible to place items on the TList outside these limits, the "isValid" function will return false in this case. The min, max and list of item names are defined when the TList variable is created (generally by a RESPONSE, PARAMETER or VARIABLE command). The presence of an item on a TList may vary during execution of the MTD. Functions exist for placing items on the TList and for various tests.

The format for a TList in a Java type cell is:

```
TList min,max,item1,item2,...,itemN
```

where:

- **min** is a number indicating the minimum number of items that can be on the list and still have it considered "valid".
- **max** is a number indicating the maximum number of items that can be on the list and still have it considered "valid".
- **name1,...,nameN** are names of items that can be placed on the list. Usually these are short names like "Male" or "HBP". (See below for an extension of this.)

Example of TList use:

```
RESPONSE GenderList  TList 0,1,1,Male,Female
```

The TList "GenderList" must have a minimum and maximum of one item, either "Male" or "Female". Note that it is possible that GenderList will have neither Male or Female, or that it will have both. In these cases, "isValid(GenderList)" will return false.

Another example:

```
VARIABLE DiseaseList  TList 0,10,HBP,HA,ASM,HiChol,CHF,ANG,OHD,DM,Stroke,TIA
```

DiseaseList can have 0 to 10 items.

There are a number of built-in functions for working with TLists. See the section on page 20. For example, to place HBP on DiseaseList, use:

```
add(DiseaseList, "HBP");
```

or

```
set(DiseaseList, "HBP", true);
```

To remove an item,
remove(DiseaseList, "HBP");

or

set(DiseaseList, "HBP", false);

Note that you can pass a "boolean expression" (just like you use in conditions) as the third argument to the set() function. For example:

set(DiseaseList, "HBP", DistolicPressure>140);

Another important function for TLists is "toString()" which is used for printing (e.g. in SS substitution). The toString function formats the names on the list using commas and the word "and". The format depends on the number of items on the list:

- 0 value (empty string)
- 1 value "a"
- 2 values "a, b and c"
- 3 values "a, b, c, and d"
- etc.

Here are some example using the DiseaseList variable above:

<table>
<thead>
<tr>
<th>DiseaseList contains</th>
<th>toString(DiseaseList) returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>nothing</td>
<td>&quot;&quot; (empty string)</td>
</tr>
<tr>
<td>HBP</td>
<td>&quot;HBP&quot;</td>
</tr>
<tr>
<td>HBP HA</td>
<td>&quot;HBP and HA&quot;</td>
</tr>
<tr>
<td>HBP HA Stroke</td>
<td>&quot;HBP, HA, and Stroke&quot;</td>
</tr>
<tr>
<td>HBP HA Stroke TIA</td>
<td>&quot;HBP, HA, Stroke, and TIA&quot;</td>
</tr>
</tbody>
</table>

The names on a TList definition can be extended to include a "long name" for each item. Instead of writing "HBP", the item can be written "HBP/High Blood Pressure". The short version before the "/" is used for normal operations. However, when a TList variable is printed (or the toString() function is used), the long names will be used. Consider:

<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>DiseaseList</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TList 0,10,HBP/high blood pressure,HA/heart attack,ASM/asthma,HICol/high cholesterol,chf/congestive heart failure,ANG/angina,OH/D/other heart disease,DM/Stoke/stroke,TIA</td>
</tr>
</tbody>
</table>

In this case:

<table>
<thead>
<tr>
<th>DiseaseList contains</th>
<th>toString(DiseaseList) returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>nothing</td>
<td>&quot;&quot; (empty string)</td>
</tr>
<tr>
<td>HBP</td>
<td>&quot;high blood pressure&quot;</td>
</tr>
<tr>
<td>HBP HA</td>
<td>&quot;high blood pressure and heart attack&quot;</td>
</tr>
<tr>
<td>HBP HA Stroke</td>
<td>&quot;high blood pressure, heart attack, and stroke&quot;</td>
</tr>
<tr>
<td>HBP HA Stroke TIA</td>
<td>&quot;high blood pressure, heart attack, stroke, and TIA&quot;</td>
</tr>
</tbody>
</table>

**Embedded Styles**

When a TCD file is "Published", a "layout file" is used to define the format of the published document (in PostScript for printing or HTML for Web). The layout file defines a number of "Styles" which are used throughout the document. Generally, these styles are applied to whole sections by the layout file.

As an extension to this, it is possible to specify a layout style in the MTD file text. This is very powerful because the style can be specified using a logical name and the actual choice of font, size, etc. is specified in the layout file.

To specify that a range of text is to have a particular style, surround it with `%style` commands. Start with `%style name%` where name is the style name as defined in the layout file. End with either another style name or `%style%` to return to the default style.

This is admittedly clumsy. You may use %s as an abbreviation for `%style`. In addition, it may be possible in the future to support Word styles automatically. (This is not possible using Word 97 as it does not save style information into HTML files.)

For example (underscore added for clarity):

<table>
<thead>
<tr>
<th>TEXT</th>
<th><code>%style header%Introduction%style%</code> The layout file has a style named &quot;header&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEXT</td>
<td>This booklet was designed using computer Use the &quot;emphasis&quot; style</td>
</tr>
</tbody>
</table>
Note: The embedded style feature is currently only available for PostScript publishing. Support can be added for HTML in the future if there is a need.

**Min/Max Options**

It is frequently necessary for content developers to estimate length of minimum and maximum versions of a document. The MTD format allows the specification of "min" and/or "max" in the options column for GROUP, LEVEL, TEXT, GRAPHIC, CONTENT, TABLE and CHART. When using Grinder, it will be possible to select "min" or "max" and create a document based solely on these setting, ignoring the condition expressions.

See the TEXT Command section on page 13 for an example of using min and max.

**Tables and Charts**

New commands allow the creation of tailored tabular data that can then be formatted into tables or charts by the appropriate publisher (PostScript or HTML). Note that these commands merely supply the tailored data to be used and the layout file defines the format of the table or chart.

See the TABLE and CHART commands for more information.

**Bulleted Lists**

To create a bulleted list in content text, use the standard MS Word bullet facility. For example:

```
/TEXT

* This is the first line of a bulleted list.
* This is the second line. It has much more text and as a result, it wraps around.
* This is the third line.
```

Text with this format will be published appropriately. In the PostScript case, the bullet character, the bullet indent and the text indent can all be specified in the layout file. In the HTML case, the `<UL>` and `<LI>` tags will be used so that the formatting will be done by the browser.

**Command Definitions**

Each of the commands is more fully explained below.

**DOCUMENT Command**

There is one DOCUMENT command and it must come before all other commands except INCLUDE, COMMENT and COMMAND.

<table>
<thead>
<tr>
<th>Command</th>
<th>Second column</th>
<th>Third column</th>
<th>Fourth Column</th>
<th>Fifth Column</th>
<th>Subsequent columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOCUMENT</td>
<td>Name (required)</td>
<td>Description (required)</td>
<td>Ignored</td>
<td>Ignored</td>
<td>Ignored</td>
</tr>
</tbody>
</table>

where:

Name is a simple name describing this project/MTD. This must be the same name as is used for the Java output file when running Grinder.

Description is a brief description of the project. It can contain spaces.

Example:

| DOCUMENT | ZAP2 | ZYBAN Advantage Plan |

Page 9 of 30
**RESPONSE Command**

During the development of a questionnaire, the MTD format is used to define response information. For each discrete response in the questionnaire, a simple variable is defined using the new RESPONSE command.

<table>
<thead>
<tr>
<th>Command</th>
<th>Second column</th>
<th>Third column</th>
<th>Fourth column</th>
<th>Fifth column</th>
<th>Sixth column</th>
<th>Remaining Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESPONSE</td>
<td>Name (required)</td>
<td>Options</td>
<td>Type</td>
<td>Initialization/Validation code</td>
<td>Label</td>
<td>Comments</td>
</tr>
</tbody>
</table>

where:

- **Name** is a simple name describing this response variable. It becomes a Java variable name that can be referenced in subsequent Java code such as characterization or conditions.
- **Options** is a list of keywords that specify optional items. Currently only one is defined for the RESPONSE command:
  - req or required this response is required (may not be left blank)
- **Type** is a Java type. Common examples are:
  - int (e.g. 1, 2, 3)
  - float (floating-point number e.g. 3.14159)
  - boolean (has two values, true and false)
  - Date (contains date/time information)
  - String (contains a text string)
- **Initialization/Validation code** is a Java snippet that does initialization after the data is imported and also checks whether the response data is valid.
- **Label** This text is used in the Grinder user interface to prompt the user. If not included, just the response name is used.

Example:

<table>
<thead>
<tr>
<th>Command</th>
<th>Second column</th>
<th>Third column</th>
<th>Fourth column</th>
<th>Fifth column</th>
<th>Remaining Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESPONSE</td>
<td>YearsSmoked</td>
<td>req</td>
<td>int</td>
<td>YearsSmokedValid = YearsSmoked&gt;0 &amp;&amp; YearsSmoked&lt;100;</td>
<td>How many years have you smoked?</td>
</tr>
<tr>
<td>RESPONSE</td>
<td>Gender</td>
<td>req</td>
<td>String</td>
<td>GenderValid = stringsEqual(Gender,&quot;Male&quot;)</td>
<td></td>
</tr>
</tbody>
</table>

**PARAMETER Command**

The PARAMETER command defines a variable which may be used by other variables and in conditions. Parameters are read from the database at the beginning of tailoring and are written back to the database at the end.

<table>
<thead>
<tr>
<th>Command</th>
<th>Second column</th>
<th>Third column</th>
<th>Fourth column</th>
<th>Fifth column</th>
<th>Remaining Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARAMETER</td>
<td>Name</td>
<td>Options</td>
<td>Type</td>
<td>Characterization code</td>
<td>Comments</td>
</tr>
</tbody>
</table>

where:

- **Name** is a simple name describing this characterization variable. It becomes a Java variable name that can be referenced in subsequent Java code such as characterizations or conditions.
- **Options** is a list of keywords that specify optional items. Currently, none are defined for this command.
- **Type** is a Java type. See the RESPONSE command.

Characterization code is a Java snippet that sets the value of the PARAMETER variable. It can reference RESPONSE variables as well as other characterization variables already defined (occurring earlier in the MTD).

Example:

```java
PARAMETER NewMaleSmkr boolean NewMaleSmkr = YearsSmoked<2 && stringsEqual(Gender,"Male");
PARAMETER NewFemaleSmkr boolean if (YearsSmoked<2 && stringsEqual(Gender,"Female")) NewFemaleSmkr = true; else
```
**VARIABLE Command**

The new VARIABLE command is essentially the same as the PARAMETER command. The only difference is that VARIABLE variables are not written to the database in production.

<table>
<thead>
<tr>
<th>Command</th>
<th>Second column</th>
<th>Third column</th>
<th>Forth column</th>
<th>Fifth column</th>
<th>Remaining Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARIABLE</td>
<td>Name</td>
<td>Options</td>
<td>Type</td>
<td>Characterization code</td>
<td>Comments</td>
</tr>
</tbody>
</table>

where:

(see PARAMETER)

Example:

```
VARIABLE isFemale boolean ifFemale = stringsEqual(Gender,"Female")
VARIABLE isMale boolean isMale = !isFemale
```

**GROUP Command**

This command defines the beginning of a collection of commands that are treated as a group. The condition code applies to all the commands in the group. Groups can contain other groups, either defined using GROUP or LEVEL. The list is terminated by an END command.

<table>
<thead>
<tr>
<th>Command</th>
<th>Second column</th>
<th>Third column</th>
<th>Forth column</th>
<th>Fifth column</th>
<th>Remaining Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>GROUP</td>
<td>Name</td>
<td>Options</td>
<td>Condition</td>
<td>Unused</td>
<td>Comments</td>
</tr>
</tbody>
</table>

where:

Name is a simple name describing this group. Publication layout templates use this name.

Options is a list of keywords that specify optional information.

min this group is part of the “min” configuration

max this group is part of the “max” configuration

Condition is a Java expression defining whether the group members should be selected.

Example:

```
GROUP Intro max ShowIntro=true Define Intro group
TEXT IntroText Welcome to the... Some text for the Intro group
GRAPHIC IntroGraphic A graphic for the Intro group
END Intro End of Intro group

GROUP Part1 min ShowPart1=true Define Part1 Group
GROUP Part1Section1 Sub group inside Part1
TEXT Part1Section1 Text inside Part1Section1
END Part1Section1 End of Part1Section1
GROUP Part1Section2 Define another section in Part1
TEXT Part1Section2 Text for it
END Part1Section2 End of Part1Section2
END Part1 End of Part1

GROUP Part2 min.max ShowPart2=true Start of Part2 Group...
```

**END Command**

The END command terminates a group.

<table>
<thead>
<tr>
<th>Command</th>
<th>Second column</th>
<th>Third</th>
<th>Forth column</th>
<th>Fifth column</th>
<th>Remaining Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command</td>
<td>Second column</td>
<td>Third column</td>
<td>Forth column</td>
<td>Fifth column</td>
<td>Remaining Columns</td>
</tr>
<tr>
<td>---------</td>
<td>---------------</td>
<td>--------------</td>
<td>--------------</td>
<td>--------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>END</td>
<td>Name</td>
<td>Options</td>
<td>Unused</td>
<td>Unused</td>
<td>Comments</td>
</tr>
</tbody>
</table>

where:
- **Name** is the name of the GROUP that is ending. If present and it does not match the appropriate group, an error is generated.
- **Options** Currently, no options are defined for this command.

Example: See the GROUP command.

**LEVEL Command**

This command is similar to GROUP in that it defines the beginning of a collection of commands that are treated as a group. However, no END command is required. The group is terminated when another LEVEL at the same level or higher is found. (Here, a higher level means a lower level number, e.g. level 1 is higher than level 2.)

The condition code applies to all the commands in the group. Groups can contain other groups, either defined using GROUP or LEVEL.

<table>
<thead>
<tr>
<th>Command</th>
<th>Second column</th>
<th>Third column</th>
<th>Forth column</th>
<th>Fifth column</th>
<th>Remaining Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEVEL</td>
<td>Levelnum</td>
<td>Name</td>
<td>Options</td>
<td>Condition</td>
<td>Unused</td>
</tr>
</tbody>
</table>

where:
- **Levelnum** is a number. Level 1 the top level (beneath the DOCUMENT “level”).
- **Name** is a simple name describing this group. Publication layout templates use this name.
- **Options** is a list of keywords that specify optional information.
  - **min** this group is part of the “min” configuration
  - **max** this group is part of the “max” configuration
- **Condition** is a Java expression defining whether the group members should be selected.

Example:

<table>
<thead>
<tr>
<th>LEVEL 1</th>
<th>Intro</th>
<th>ShowIntro=true</th>
<th>Define Intro group</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEXT</td>
<td>IntroText</td>
<td></td>
<td>Welcome to the ..</td>
</tr>
<tr>
<td>GRAPHIC</td>
<td>IntroGraphic</td>
<td></td>
<td>Some text for the Intro group</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LEVEL 1</th>
<th>Part1</th>
<th>ShowPart1=true</th>
<th>Define Part1 group, same level</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEVEL 2</td>
<td>Part1Section1</td>
<td></td>
<td>Sub group inside Part1</td>
</tr>
<tr>
<td>TEXT</td>
<td></td>
<td></td>
<td>Text inside Part1Section1</td>
</tr>
<tr>
<td>LEVEL 2</td>
<td>Part1Section2</td>
<td></td>
<td>Define another section in Part1</td>
</tr>
<tr>
<td>TEXT</td>
<td></td>
<td></td>
<td>Text for it</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LEVEL 1</th>
<th>Part2</th>
<th></th>
<th>Back to level 1, start group</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Part2</td>
</tr>
</tbody>
</table>

A Note about GROUP and LEVEL

The content developer may use either GROUP or LEVEL. Each has advantages. The GROUP command allows groups of commands to be moved up or down in the hierarchy without editing level numbers. The LEVEL command is more explicit and does not require the END command. Further, SECTION and TOPIC may be used as synonyms for LEVEL 1 and LEVEL 2.
**TEXT Command**

The TEXT command is the primary mechanism for specifying text that may be included in the tailored output. The primary version of the text is placed in the fifth column. Alternative versions (e.g. Spanish or numeric data) may be added in column six and beyond. An external mechanism selects which text column to use.

<table>
<thead>
<tr>
<th>Command</th>
<th>Second column</th>
<th>Third column</th>
<th>Fourth column</th>
<th>Fifth column</th>
<th>Sixth column</th>
<th>Remaining Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEXT</td>
<td>Name</td>
<td>Options</td>
<td>Condition</td>
<td>Primary Text</td>
<td>Text or comments</td>
<td>Text or comments</td>
</tr>
</tbody>
</table>

where:

- Name: is a simple name describing this block of text.
- Options: is a list of keywords that specify optional information.
  - min: this text is part of the "min" configuration
  - max: this text is part of the "max" configuration
- Condition: is a Java expression defining whether this text should be selected.
- Text: is formatted text as described in the "Text Cells" section on page 5.

Example:

<table>
<thead>
<tr>
<th>Command</th>
<th>Second column</th>
<th>Third column</th>
<th>Fourth column</th>
<th>Fifth column</th>
<th>Sixth column</th>
<th>Remaining Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEXT</td>
<td>IntroHeader</td>
<td>min, max</td>
<td>Introduction</td>
<td></td>
<td>Unconditional Text with bold</td>
<td></td>
</tr>
<tr>
<td>TEXT</td>
<td>min</td>
<td>Name isKnown=false</td>
<td>Welcome to the ...</td>
<td></td>
<td>Conditional</td>
<td></td>
</tr>
<tr>
<td>TEXT</td>
<td>max</td>
<td>Name isKnown=true</td>
<td>Welcome SSFirstNameSS, ...</td>
<td></td>
<td>Conditional with substitution</td>
<td></td>
</tr>
</tbody>
</table>

**GRAPHIC Command**

The GRAPHIC command allows the specification of conditional graphics to be included in the tailored content. The GRAPHIC command conditionally defines the name of a graphic element. The layout document specifies which graphics file is to be used for this name.

<table>
<thead>
<tr>
<th>Command</th>
<th>Second column</th>
<th>Third column</th>
<th>Fourth column</th>
<th>Fifth column</th>
<th>Remaining Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRAPHIC</td>
<td>Name</td>
<td>Options</td>
<td>Condition</td>
<td>Unused</td>
<td>Comments</td>
</tr>
</tbody>
</table>

where:

- Name: is a simple name describing this graphic. Used by the layout file to specify an actual graphics file.
- Options: are a list of keywords that specify optional information.
  - min: this text is part of the "min" configuration
  - max: this text is part of the "max" configuration
- Condition: is a Java expression defining whether this graphic should be selected.

Example:

<table>
<thead>
<tr>
<th>Command</th>
<th>Second column</th>
<th>Third column</th>
<th>Fourth column</th>
<th>Fifth column</th>
<th>Sixth column</th>
<th>Remaining Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRAPHIC</td>
<td>YoungManSmoking</td>
<td>min, max</td>
<td>isMale &amp; isASmoker</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**CONTENT Command**

CONTENT allows a more general type of content. Like TEXT and GRAPHIC, it has a condition column and is often part of a GROUP. In addition, it has arbitrary Java code which can dynamically generate content.

<table>
<thead>
<tr>
<th>Command</th>
<th>Second column</th>
<th>Third column</th>
<th>Fourth column</th>
<th>Fifth column</th>
<th>Remaining Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONTENT</td>
<td>Name</td>
<td>Options</td>
<td>Condition</td>
<td>Java code</td>
<td>Comments</td>
</tr>
</tbody>
</table>

where:
**Name**

is a simple name describing this content.

**Options**

are a list of keywords that specify optional information.

- `min` this text is part of the “min” configuration
- `max` this text is part of the “max” configuration

**Condition**

is a Java expression defining whether this content should be selected.

**Java code**

is arbitrary Java code. Generally it will contain calls to “writeText” and “writeGraphic” resulting in text and graphic content being selected. It might also contain code to math or database access.

**Example:**

```
String Recipe = getDBString(RecipeNum, "recipe");
writeText(Recipe);
int Calories = getDBInt(RecipeNum, "calories");
writeText("Calories: "+ Calories);
TotalCalories += Calories;
```

In the above example, if `RecipeNum` is not -1, a database function is called to get a recipe in the form of a String. This String is output using the “writeText” function. Similarly, a calorie value is fetched and text is dynamically generated to display this value. Finally, the total calories is updated.

Note: The “getDBString” and “getDBInt” functions are used as an example and may not be implemented exactly as shown.

**SECTION Command**

The `SECTION` command is completely equivalent to “LEVEL 1”.

**TOPIC Command**

The `TOPIC` command is completely equivalent to “LEVEL 2”.

**TABLE Command**

The `TABLE` command allows the creation of tailored tabular data that can then be formatted into table form by the appropriate publisher (PostScript or HTML). Note that this command merely supplies the tailored data to be used and the layout file defines the format of the table.

<table>
<thead>
<tr>
<th>Command</th>
<th>Second column</th>
<th>Third column</th>
<th>Forth column</th>
<th>Fifth column</th>
<th>Remaining Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>SECTION</td>
<td>Name</td>
<td>Options</td>
<td>Condition</td>
<td>Unused</td>
<td>Comments</td>
</tr>
<tr>
<td>TOPIC</td>
<td>Name</td>
<td>Options</td>
<td>Condition</td>
<td>Unused</td>
<td>Comments</td>
</tr>
<tr>
<td>TABLE</td>
<td>Name</td>
<td>Options</td>
<td>Condition</td>
<td>Table data</td>
<td>Text or comments</td>
</tr>
</tbody>
</table>

**where:**

- **Name** is a simple name describing this table. The publisher will use this name to format and position the table.
- **Options** is a list of keywords that specify optional information.
  - `min` this table is part of the “min” configuration
  - `max` this table is part of the “max” configuration
- **Condition** is a Java expression defining whether this table should be selected.
- **Table data** is a list of data items, separated by commas and semicolons. Commas separate table cells within a row; semicolons separate rows. The data items must be strings without commas or delimiters. Use SS-SS for...
Java expression substitution is allowed. Spaces and paragraph breaks are ignored and may be used to make the rows more readable.

Example:

<table>
<thead>
<tr>
<th>TABLE</th>
<th>SingleRow1</th>
<th>123, 456, 789</th>
<th>Static data, 3 items in a single row</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE</td>
<td>SingleRow2</td>
<td>$\text{data1}$, $\text{data2}$, $\text{data3}$</td>
<td>Dynamic data from three variables</td>
</tr>
<tr>
<td>TABLE</td>
<td>ThreeRow</td>
<td>Male, $\text{data[0]}$; Female, $\text{data[1]}$; Total, $\text{data[2]}$</td>
<td>Static &amp; dynamic, 2 items in 3 rows</td>
</tr>
<tr>
<td>TABLE</td>
<td>ThreeRowHdr</td>
<td>Group, Count; Male, $\text{data[0]}$; Female, $\text{data[1]}$; Total, $\text{data[2]}$</td>
<td>Same with a header row</td>
</tr>
</tbody>
</table>

Possible sample output for SingleRow1 and SingleRow2 (remember, the formatting is controlled by the layout file):

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>456</td>
<td>789</td>
</tr>
</tbody>
</table>

Possible sample output for ThreeRow:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Male</td>
<td>111</td>
</tr>
<tr>
<td>Female</td>
<td>222</td>
</tr>
<tr>
<td>Total</td>
<td>333</td>
</tr>
</tbody>
</table>

Possible sample output for ThreeRowHdr:

<table>
<thead>
<tr>
<th>Group</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Male</td>
<td>111</td>
</tr>
<tr>
<td>Female</td>
<td>222</td>
</tr>
<tr>
<td>Total</td>
<td>333</td>
</tr>
</tbody>
</table>

**CHART Command**

The CHART command allows the creation of tailored tabular data which can then be displayed in chart form by the appropriate publisher (PostScript or HTML). Note that this command merely supplies the data to be used and the layout file defines the format of the chart.

The CHART command is essentially identical to the TABLE command and results in the same type of data being passed to the publisher.

<table>
<thead>
<tr>
<th>Command</th>
<th>Second column</th>
<th>Third column</th>
<th>Forth column</th>
<th>Fifth column</th>
<th>Remaining Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHART</td>
<td>Name</td>
<td>Options</td>
<td>Condition</td>
<td>Table data</td>
<td>Text or comments</td>
</tr>
</tbody>
</table>

where:

- **Name** is a simple name describing this chart. The publisher will use this name to format and place the chart.
- **Options** is a list of keywords that specify optional information.
  - min  this chart is part of the "min" configuration
  - max  this chart is part of the "max" configuration
- **Condition** is a Java expression defining whether this chart should be selected.
Table data is a list of data items, separated by commas and semicolons. Commas separate table cells within a row; semicolons separate rows. The data items must be strings without commas or delimiters. Use $SS-SS$ for Java expression substitution is allowed. Spaces and paragraph breaks are ignored and may be used to make the rows more readable.

General strings can be passed as labels in the first item of each row. The rest of the data values must evaluate to a numeric value.

Example:

<table>
<thead>
<tr>
<th>CHART</th>
<th>SingleSet1</th>
<th>123,456,789</th>
<th>Static data, 3 numeric data values</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHART</td>
<td>SingleRow2</td>
<td>$SSdata1$$, $SSdata2$$, $SSdata3$$</td>
<td>Dynamic data from three numeric variables in a single row</td>
</tr>
<tr>
<td>CHART</td>
<td>ThreeRowLabel</td>
<td>Male, $SSx1[0]$$, $SSx1[1]$$; Female, $SSx2[0]$$, $SSx2[1]$$; Total, $SSx3[0]$$, $SSx3[0]$$</td>
<td>The first item in each row contains label for that dataset. x1, x2 and x3 are arrays containing datasets.</td>
</tr>
</tbody>
</table>

Possible sample output for SingleSet1 and SingleSet2 (remember, the formatting is controlled by the layout file):

![Possible sample output for SingleSet1 and SingleSet2](image)

Possible sample output for ThreeRowLabel:

![Possible sample output for ThreeRowLabel](image)

**SENTENCE Command**

The SENTENCE is completely equivalent to “TEXT”.

Page 16 of 30
**FUNCTION Command**

The new **FUNCTION** command allows the definition of a Java function that may be used in subsequent expressions including conditions.

<table>
<thead>
<tr>
<th>Command</th>
<th>Second column</th>
<th>Third column</th>
<th>Forth column</th>
<th>Fifth column</th>
<th>Sixth column</th>
<th>Remaining Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>FUNCTION</td>
<td>Name</td>
<td>Options</td>
<td>Condition</td>
<td>Primary Text</td>
<td>Text or comments</td>
<td>Text or comments</td>
</tr>
</tbody>
</table>

where:

- **Name** is a simple name for the function.
- **Options** is a list of keywords that specify optional information. Currently, none are defined for this command.
- **Type** is a Java type. See the **RESPONSE** command.
- **Java code** is Java code that defines the function.

Example:

```java
FUNCTION isTuesday boolean Calendar cal = Calendar.getInstance();
    boolean isTues = cal.get(Calendar.DAY_OF_WEEK) == Calendar.TUESDAY;
    return isTues;
```

In the above example, a Java function is defined which returns a boolean value.

**JAVA Command**

The **JAVA** command allows the inclusion of raw Java code. This is an advanced feature and will often not be used.

<table>
<thead>
<tr>
<th>Command</th>
<th>Second column</th>
<th>Third column</th>
<th>Forth column</th>
<th>Fifth column</th>
<th>Remaining Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>JAVA</td>
<td>unused</td>
<td>Options</td>
<td>unused</td>
<td>Raw Java code</td>
<td>Comments</td>
</tr>
</tbody>
</table>

where:

- **Options** are a list of keywords that specify optional information. Currently, none are defined for this command.
- **Java code** is Java code that will be included in the generated source file.

Example:

```java
JAVA
    Calendar cal = Calendar.getInstance();
    boolean isTuesday () {
        return cal.get(Calendar.DAY_OF_WEEK) == Calendar.TUESDAY;
    }
```

The Java code above creates a global variable named "cal" which contains a Calendar object. The function "isTuesday" is also defined and it can be called from any Java code.

**INCLUDE Command**

This command allows a MTD to be broken up into smaller files. Typically, a primary MTD file will include sub-files, each of which may be developed separately, perhaps by a different content developer.

<table>
<thead>
<tr>
<th>Command</th>
<th>Second column</th>
<th>Third column</th>
<th>Forth column</th>
<th>Fifth column</th>
<th>Remaining Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>INCLUDE</td>
<td>FilePathName</td>
<td>Options</td>
<td>Unused</td>
<td>Unused</td>
<td>Comments</td>
</tr>
</tbody>
</table>

---

Page 17 of 30
where:

FilePathName is the pathname of another MTD file in HTML format. The path can be either relative (in the same folder or lower) or absolute (a full pathname).

Options Currently, no options are defined for this command.

Example:

<table>
<thead>
<tr>
<th>Command</th>
<th>Second column</th>
<th>Third column</th>
<th>Forth column</th>
<th>Fifth column</th>
<th>Remaining Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>INCLUDE</td>
<td>ZAP2Responses.html</td>
<td></td>
<td></td>
<td></td>
<td>Relative, in same folder</td>
</tr>
<tr>
<td>INCLUDE</td>
<td>Current/ZAP2Responses.html</td>
<td></td>
<td></td>
<td></td>
<td>Relative, in “Current” folder</td>
</tr>
<tr>
<td>INCLUDE</td>
<td>P:\ZAP2\MTD\ZAP2Responses.html</td>
<td></td>
<td></td>
<td></td>
<td>Absolute, on a server</td>
</tr>
</tbody>
</table>

**SCHEDULE Command**

The SCHEDULE command is used to determine when a document is to be executed. The Java code is executed everyday by the production system until it indicates that the document is ready to be run. Once run, the production system will subsequently not call the Java code.

<table>
<thead>
<tr>
<th>Command</th>
<th>Second column</th>
<th>Third column</th>
<th>Forth column</th>
<th>Fifth column</th>
<th>Remaining Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCHEDULE</td>
<td>Unused</td>
<td>Options</td>
<td>Unused</td>
<td>Condition Java code</td>
<td>Comments</td>
</tr>
</tbody>
</table>

where:

Options Currently, no options are defined for this command.

Condition Java Code Java code that evaluates to a boolean value.

Example:

<table>
<thead>
<tr>
<th>Command</th>
<th>Second column</th>
<th>Third column</th>
<th>Forth column</th>
<th>Fifth column</th>
<th>Remaining Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCHEDULE</td>
<td></td>
<td></td>
<td></td>
<td>dateToDay(today()) &gt; (dateToDay(QuitDate)+14)</td>
<td></td>
</tr>
</tbody>
</table>

This Java code will evaluate to “true” when the “today()” function returns a date which is greater than 14 days after QuitDate.

**COMMENT Command**

Lines beginning with COMMENT are completely ignored.

<table>
<thead>
<tr>
<th>Command</th>
<th>Second column</th>
<th>Third column</th>
<th>Forth column</th>
<th>Fifth column</th>
<th>Remaining Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMMENT</td>
<td>Comments</td>
<td>Comments</td>
<td>Comments</td>
<td>Comments</td>
<td>Comments</td>
</tr>
</tbody>
</table>

**COMMAND and VERSION Commands**

Lines beginning with COMMAND or VERSION are completely ignored. The COMMAND command may be used to label columns. VERSION may be useful for recording modifications to the MTD file.

<table>
<thead>
<tr>
<th>Command</th>
<th>Second column</th>
<th>Third column</th>
<th>Forth column</th>
<th>Fifth column</th>
<th>Remaining Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMMAND</td>
<td>Ignored</td>
<td>Ignored</td>
<td>Ignored</td>
<td>Ignored</td>
<td>Ignored</td>
</tr>
<tr>
<td>VERSION</td>
<td>Ignored</td>
<td>Ignored</td>
<td>Ignored</td>
<td>Ignored</td>
<td>Ignored</td>
</tr>
</tbody>
</table>

Example:

<table>
<thead>
<tr>
<th>Command</th>
<th>Name</th>
<th>Options</th>
<th>Conditions</th>
<th>Text</th>
<th>Text or comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEVEL 1</td>
<td>Intro</td>
<td>ShowIntro=true</td>
<td></td>
<td></td>
<td>Define Intro group</td>
</tr>
<tr>
<td>TEXT</td>
<td>IntroText</td>
<td></td>
<td>Welcome to the Intro group</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GRAPHIC</td>
<td>IntroGraphic</td>
<td></td>
<td></td>
<td>A graphic for the Intro group</td>
<td></td>
</tr>
</tbody>
</table>

**LABEL Command**

A LABEL command provides a string of text used in a user interface. In particular, the Grinder application displays the label when it builds its user interface for inputting responses.

<table>
<thead>
<tr>
<th>Command</th>
<th>Second column</th>
<th>Third column</th>
<th>Forth column</th>
<th>Fifth column</th>
<th>Sixth Column</th>
</tr>
</thead>
</table>

Page 18 of 30
<table>
<thead>
<tr>
<th>LABEL</th>
<th>Ignored</th>
<th>Ignored</th>
<th>Ignored</th>
<th>Ignored</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>LABEL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Personal Information</td>
</tr>
<tr>
<td>RESPONSE</td>
<td>FirstName</td>
<td>String</td>
<td></td>
<td></td>
<td>Please enter your first name</td>
</tr>
</tbody>
</table>

In this example, Grinder will display “Personal Information” on the line before it prompts for FirstName.
### Built-In Functions

The following is a list of functions available from any Java code. Many more functions will be added.

Some examples marked * because their use is limited and a good example requires more space than provided here.

There are three kinds of examples shown below:

- Content text with `SS-SS` substitution where a function is used to insert information into the text. E.g.:
  
  Your birthdate is SSS(String(BirthDate, DateFormat.LONG)S).

- Examples with semicolons indicating complete Java statements that could be used to define a VARIABLE. E.g.:
  
  ```java
  IsSmoker = toBoolean(Other);  
  ```

- Examples without semicolons are Java boolean expressions that could be used in a tailoring condition. E.g.:

  ```java
  equal(FirstName, "Neil")
  ```

### Preferred String Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Returns</th>
<th>Example</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>equal(String s1, String s2)</td>
<td>boolean</td>
<td>equal(FirstName, &quot;Neil&quot;)</td>
<td>Return true if strings are equal, ignoring case.</td>
</tr>
<tr>
<td>length(String s)</td>
<td>int</td>
<td>length(FirstName)</td>
<td>Returns the length of a string.</td>
</tr>
<tr>
<td>toBoolean(String s)</td>
<td>boolean</td>
<td>IsSmoker = toBoolean(Other);</td>
<td>Converts a String into a boolean. Returns true of str is &quot;1&quot;, &quot;y&quot;, &quot;yes&quot;, &quot;t&quot; or &quot;true&quot; (case does not matter), false otherwise.</td>
</tr>
<tr>
<td>toInt(String s)</td>
<td>int</td>
<td>YearsSmoked = toInt(YearsSmoked);</td>
<td>Converts a String into an int.</td>
</tr>
<tr>
<td>toFloat(String)</td>
<td>float</td>
<td>Weight = toFloat(Weight);</td>
<td>Converts a String into a float value.</td>
</tr>
<tr>
<td>toDate(String)</td>
<td>Date</td>
<td>BirthDate = toDate(BirthDateStr);</td>
<td>Converts a String into a Date value.</td>
</tr>
<tr>
<td>toString(Boolean b)</td>
<td>String</td>
<td>*</td>
<td>Converts a boolean to a String, either &quot;true&quot; or &quot;false&quot;.</td>
</tr>
<tr>
<td>toString(int i)</td>
<td>String</td>
<td>*</td>
<td>Converts an int to a String.</td>
</tr>
<tr>
<td>toString(float f)</td>
<td>String</td>
<td>*</td>
<td>Converts a float to a String.</td>
</tr>
<tr>
<td>toString(Date d)</td>
<td>String</td>
<td>*</td>
<td>Converts a Date to a String using the short format (e.g. 1/12/52).</td>
</tr>
<tr>
<td>toString(Date d, int format)</td>
<td>String</td>
<td>Your birthdate is SSS(String(BirthDate, DateFormat.LONG)S);</td>
<td>Converts a Date to a String specifying a format. Legal formats are: DateFormat.SHORT 1/12/52, DateFormat.LONG January 12, 1952, DateFormat.FULL Tuesday, January 12, 1952</td>
</tr>
</tbody>
</table>

### Preferred Date and and Day Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Returns</th>
<th>Example</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>toDay(Date d)</td>
<td>int</td>
<td>QuitDays = thisDay() - toDay(QuiDate);</td>
<td>Converts a Date to an int containing the number of days since 1/1/70. Dates before then have negative numbers. The example computes the number of days between today and the date in the QuitDate variable.</td>
</tr>
<tr>
<td>toDate(int day)</td>
<td>Date</td>
<td>EmailDate =</td>
<td>Converts a day number to a date. The example adds 10 to the current day number and converts</td>
</tr>
</tbody>
</table>

Page 20 of 30
<table>
<thead>
<tr>
<th>Function</th>
<th>Returns</th>
<th>Example</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>thisDate()</td>
<td>Date</td>
<td>equal(thisDate(), EmailDate)</td>
<td>Returns a Date value for today's date.</td>
</tr>
<tr>
<td>thisDay()</td>
<td>int</td>
<td>See toDay and toDate</td>
<td>Day number for today.</td>
</tr>
<tr>
<td>equal(Date d1, Date d2)</td>
<td>boolean</td>
<td>equal(thisDate(), EmailDate)</td>
<td>Returns true if the two dates reference the same day.</td>
</tr>
</tbody>
</table>

**TList Functions:**

<table>
<thead>
<tr>
<th>Function</th>
<th>Returns</th>
<th>Example</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>isValid(TList list)</td>
<td>boolean</td>
<td>isValid(DiseaseList)</td>
<td>Returns true if the number of items in the list are within the range specified by the min and max.</td>
</tr>
<tr>
<td>toString(TList list)</td>
<td>String</td>
<td>*</td>
<td>Converts the list to a comma separated list of values, its format depending on the number of items in the list: 0 (empty string) 1 a 2 a and b 3 a, b, and c 4 a, b, c, and d etc...</td>
</tr>
<tr>
<td>toTList(TList list, String str)</td>
<td></td>
<td>toTList(DiseaseList, &quot;HA,HBP,ASM&quot;);</td>
<td>Loads items on the list from a string.</td>
</tr>
<tr>
<td>isEmpty(TList list)</td>
<td>boolean</td>
<td>isEmpty(DiseaseList)</td>
<td>Returns true if there are no items on the list.</td>
</tr>
<tr>
<td>count(TList list)</td>
<td>int</td>
<td>count(DiseaseList)&gt;3</td>
<td>Returns the number of items on the list.</td>
</tr>
<tr>
<td>set(TList list, String name, boolean present)</td>
<td></td>
<td>set(DiseaseList, &quot;ASM&quot;, AsthmaScore&gt;3);</td>
<td>If &quot;present&quot; evaluates to true, places named item on the list. If false, removes it.</td>
</tr>
<tr>
<td>add(TList list, String name)</td>
<td></td>
<td>add(DiseaseList, &quot;ASM&quot;);</td>
<td>Places the named item on the list unconditionally.</td>
</tr>
<tr>
<td>remove(TList list, String name)</td>
<td></td>
<td>remove(DiseaseList, &quot;ASM&quot;);</td>
<td>Removes the named item from the list unconditionally.</td>
</tr>
<tr>
<td>on(TList list, String name)</td>
<td>boolean</td>
<td>on(DiseaseList, &quot;ASM&quot;);</td>
<td>Returns true if the named item is on the list.</td>
</tr>
<tr>
<td>onTop(TList list, String name)</td>
<td>boolean</td>
<td>onTop(DiseaseList, &quot;ASM&quot;);</td>
<td>Returns true if the named item is on the top of the list.</td>
</tr>
<tr>
<td>inTop(TList list, String name, int n)</td>
<td>boolean</td>
<td>InTop(DiseaseList, &quot;ASM&quot;, 3)</td>
<td>Creates a new TList and returns it. Only the names on &quot;list&quot; between &quot;start&quot; and &quot;stop&quot; are transferred to the new list. The order of those names is not changed. &quot;start&quot; and &quot;stop&quot; are numbers, where 1 indicates the first item on the list. In the example, &quot;Top3List&quot; will contain 0, 1, 2 or 3 items, depending what items are in the top three slots of &quot;DiseaseList&quot;.</td>
</tr>
<tr>
<td>range(TList list, int start, int stop)</td>
<td>TList</td>
<td>Top3List = range(DiseaseList, 1, 3)</td>
<td>Returns true if &quot;name&quot; is on &quot;list&quot; in the range specified by &quot;start&quot; and &quot;stop&quot;, where 1 indicates the first item on the list. In the example, returns true if ASM is the second or third item on the list.</td>
</tr>
<tr>
<td>Function</td>
<td>Returns</td>
<td>Example</td>
<td>Explanation</td>
</tr>
<tr>
<td>--------------------------</td>
<td>---------</td>
<td>----------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>length(TList list)</td>
<td>int</td>
<td>*</td>
<td>Returns the size of the list, that is, the number of names that were defined when the list was created.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>In each of the following, “names” contains a comma-separated list of names (e.g. “a,b,d”).</td>
</tr>
<tr>
<td>containsAll(TList list,</td>
<td>boolean</td>
<td>containsAll(DiseaseList,</td>
<td>Returns true if all the items in “names” are on the list. The list might also contain other names.</td>
</tr>
<tr>
<td>String names)</td>
<td></td>
<td>&quot;ASM,HD,HBP&quot;)</td>
<td></td>
</tr>
<tr>
<td>containsSome(TList list,</td>
<td>boolean</td>
<td>containsSome(DiseaseList,</td>
<td>Returns true if at least one of the items in “names” is on the list.</td>
</tr>
<tr>
<td>String names)</td>
<td></td>
<td>&quot;ASM,HD,HBP&quot;)</td>
<td></td>
</tr>
<tr>
<td>containsOnly(TList list,</td>
<td>boolean</td>
<td>containsOnly(DiseaseList,</td>
<td>Returns true if only the items in “names” are on the list.</td>
</tr>
<tr>
<td>String names)</td>
<td></td>
<td>&quot;ASM,HD,HBP&quot;)</td>
<td></td>
</tr>
<tr>
<td>reorder(TList list, names)</td>
<td>TList</td>
<td>PriorityList = reorder(DiseaseList,</td>
<td>Creates a new TList and returns it. Only names listed in “names” are transferred to the new list.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;ASM,HD,HBP&quot;);</td>
<td>The order of the items supported by the new list is also determined by “names”.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>or</td>
<td>When used in text content, returns a string like “toString” above.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Some of your diseases are:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>S$reorder(DiseaseList,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;ASM,HD,HBP&quot;)$$.</td>
<td></td>
</tr>
<tr>
<td>generate(TList list, names)</td>
<td>String</td>
<td>*</td>
<td>Similar to “reorder” but it returns a String instead of a TList.</td>
</tr>
</tbody>
</table>

### Misc Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Returns</th>
<th>Example</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>count(Boolean b1, ... bx)</td>
<td>int</td>
<td>SymptomCount = count(HasCough, HasSoreThroat, HasSniffles);</td>
<td>Counts the number of true values in “b1” through “bx” where x can be up to 10.</td>
</tr>
<tr>
<td>fullStateName(String name)</td>
<td></td>
<td>You live in %% fullStateName(State)%%</td>
<td>Converts a short state name (“MI”) to a long name (“Michigan”).</td>
</tr>
<tr>
<td>shortStateName(String name)</td>
<td></td>
<td>*</td>
<td>Converts a full state name (“Michigan”) to a short name (“MI”).</td>
</tr>
<tr>
<td>writeText(String, text)</td>
<td></td>
<td>*</td>
<td>Write arbitrary text to the tailored output.</td>
</tr>
<tr>
<td>writeGraphic(String name)</td>
<td></td>
<td>*</td>
<td>Writes the name of a graphic to the tailored output.</td>
</tr>
</tbody>
</table>

### Older Functions:

<table>
<thead>
<tr>
<th>Function</th>
<th>Returns</th>
<th>Example</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>stringToInt(String s)</td>
<td>int</td>
<td>*</td>
<td>Older version of toInt.</td>
</tr>
<tr>
<td>stringToBoolean(String s)</td>
<td>boolean</td>
<td>*</td>
<td>Older version of toBoolean.</td>
</tr>
<tr>
<td>stringToFloat(String s)</td>
<td>float</td>
<td>*</td>
<td>Older version of toFloat.</td>
</tr>
<tr>
<td>stringToDate(String s)</td>
<td>Date</td>
<td>*</td>
<td>Older version of toDate.</td>
</tr>
<tr>
<td>intToString(int i)</td>
<td>String</td>
<td>*</td>
<td>Older version of toString.</td>
</tr>
<tr>
<td>booleanToString(boolean b)</td>
<td>String</td>
<td>*</td>
<td>Older version of toString.</td>
</tr>
<tr>
<td>floatToString(float f)</td>
<td>String</td>
<td>*</td>
<td>Older version of toString.</td>
</tr>
<tr>
<td>dateToString(Date d)</td>
<td>String</td>
<td>*</td>
<td>Older version of toString.</td>
</tr>
<tr>
<td>Function</td>
<td>Return Type</td>
<td>Parameters</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-------------</td>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>dateToLongString(Date d)</td>
<td>String</td>
<td>*</td>
<td>Converts a Date to a String using the LONG format (January 12, 1952).</td>
</tr>
<tr>
<td>dateToFullString(Date d)</td>
<td>String</td>
<td>*</td>
<td>Converts a Date to a String using the FULL format (Tuesday, April 12, 1952).</td>
</tr>
<tr>
<td>dateToDay(Date d)</td>
<td>int</td>
<td>*</td>
<td>Converts a Date to an int containing the number of days since 1/1/70. Dates before then have negative numbers.</td>
</tr>
<tr>
<td>dayToDate(int day)</td>
<td>Date</td>
<td>*</td>
<td>Converts a day number to a date.</td>
</tr>
<tr>
<td>today()</td>
<td>Date</td>
<td>*</td>
<td>Returns a Date value for today’s date.</td>
</tr>
<tr>
<td>todayDay()</td>
<td>int</td>
<td>*</td>
<td>Returns a day number for today’s date.</td>
</tr>
<tr>
<td>datesEqual(Date d1, Date d2)</td>
<td>boolean</td>
<td>*</td>
<td>Returns true if both Dates refer to the same day. Equivalent to “dateToDay(d1)==dateToDay(d2)”. Do not use “d1==d2”.</td>
</tr>
<tr>
<td>countOfBoolean(boolean b0, boolean b1,...)</td>
<td>int</td>
<td>*</td>
<td>Counts the number of true boolean values. Up to 10 booleans are supported.</td>
</tr>
<tr>
<td>stringLength(String s)</td>
<td>int</td>
<td>*</td>
<td>Returns the length of the string.</td>
</tr>
<tr>
<td>stringsEqual(String s1, String s2)</td>
<td>boolean</td>
<td>*</td>
<td>Returns true if both strings have the same contents. Do not use “s1==s2”.</td>
</tr>
</tbody>
</table>
Special Characters

All of the characters listed below may be used. The shaded areas indicate characters which require special handling.

### Standard ASCII Special Characters

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
<th>Code in MTD as</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>Exclamation point</td>
<td>!</td>
</tr>
<tr>
<td>&quot;</td>
<td>Quotation mark (straight)</td>
<td>&quot;</td>
</tr>
<tr>
<td>#</td>
<td>Pound</td>
<td>#</td>
</tr>
<tr>
<td>$</td>
<td>Dollar</td>
<td>$</td>
</tr>
<tr>
<td>%</td>
<td>Percent</td>
<td>%</td>
</tr>
<tr>
<td>&amp;</td>
<td>Ampersand</td>
<td>&amp;</td>
</tr>
<tr>
<td>'</td>
<td>Single quote (straight)</td>
<td>'</td>
</tr>
<tr>
<td>(</td>
<td>Left paren</td>
<td>(</td>
</tr>
<tr>
<td>)</td>
<td>Right paren</td>
<td>)</td>
</tr>
<tr>
<td>*</td>
<td>Asterisk</td>
<td>*</td>
</tr>
<tr>
<td>+</td>
<td>Plus</td>
<td>+</td>
</tr>
<tr>
<td>,,</td>
<td>Comma</td>
<td>,</td>
</tr>
<tr>
<td>-</td>
<td>Minus</td>
<td>-</td>
</tr>
<tr>
<td>.</td>
<td>Period</td>
<td>.</td>
</tr>
<tr>
<td>/</td>
<td>Slash</td>
<td>/</td>
</tr>
<tr>
<td>:</td>
<td>Colon</td>
<td>:</td>
</tr>
<tr>
<td>;</td>
<td>Semicolon</td>
<td>;</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
<td>&lt;</td>
</tr>
<tr>
<td>=</td>
<td>Equals</td>
<td>=</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
<td>&gt;</td>
</tr>
<tr>
<td>?</td>
<td>Question</td>
<td>?</td>
</tr>
<tr>
<td>[</td>
<td>Left bracket</td>
<td>[</td>
</tr>
<tr>
<td>\</td>
<td>Back slash</td>
<td>\</td>
</tr>
<tr>
<td>]</td>
<td>Right bracket</td>
<td>]</td>
</tr>
<tr>
<td>^</td>
<td>Caret</td>
<td>^</td>
</tr>
<tr>
<td>_</td>
<td>Underscore</td>
<td>_</td>
</tr>
<tr>
<td>`</td>
<td>Left single quote</td>
<td>`</td>
</tr>
<tr>
<td>{</td>
<td>Left brace</td>
<td>{</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Vertical bar</td>
</tr>
<tr>
<td>}</td>
<td>Right brace</td>
<td>}</td>
</tr>
<tr>
<td>~</td>
<td>Tilde</td>
<td>~</td>
</tr>
</tbody>
</table>

### Other Special Characters

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
<th>Code in MTD as</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ω</td>
<td>Cap omega</td>
<td>#SYM##Ω##STD##</td>
</tr>
<tr>
<td>δ</td>
<td>LC delta</td>
<td>#SYM##δ##STD##</td>
</tr>
<tr>
<td>ω</td>
<td>LC omega</td>
<td>#SYM##ω##STD##</td>
</tr>
<tr>
<td>†</td>
<td>Dagger</td>
<td>†</td>
</tr>
<tr>
<td>‡</td>
<td>Double Dagger</td>
<td>‡</td>
</tr>
<tr>
<td>‘</td>
<td>Circumflex</td>
<td>‘</td>
</tr>
<tr>
<td>%</td>
<td>Permil</td>
<td>%</td>
</tr>
<tr>
<td>‹</td>
<td>Less than sign</td>
<td>‹</td>
</tr>
<tr>
<td>Æ</td>
<td>Capital OE ligature</td>
<td>Æ</td>
</tr>
<tr>
<td>'</td>
<td>Left single quote</td>
<td>'#lsq##</td>
</tr>
<tr>
<td>'</td>
<td>Right single quote</td>
<td>'#rsq##</td>
</tr>
<tr>
<td>&quot;</td>
<td>Left double quote</td>
<td>'#ldq##</td>
</tr>
<tr>
<td>&quot;</td>
<td>Right double quote</td>
<td>'#rdq##</td>
</tr>
<tr>
<td>o</td>
<td>Bullet</td>
<td>o</td>
</tr>
<tr>
<td>–</td>
<td>En dash</td>
<td>'#nd##</td>
</tr>
<tr>
<td>—</td>
<td>Em dash</td>
<td>'#md##</td>
</tr>
<tr>
<td>‘</td>
<td>Tilde</td>
<td>‘</td>
</tr>
<tr>
<td>™</td>
<td>Trademark</td>
<td>™</td>
</tr>
<tr>
<td>›</td>
<td>Greater than sign</td>
<td>›</td>
</tr>
<tr>
<td>œ</td>
<td>Small oe ligature</td>
<td>œ</td>
</tr>
<tr>
<td>‹</td>
<td>Nonbreaking space</td>
<td>'#nbsp##</td>
</tr>
<tr>
<td>i</td>
<td>Inverted exclamation point</td>
<td>i</td>
</tr>
<tr>
<td>¢</td>
<td>Cent sign</td>
<td>¢</td>
</tr>
<tr>
<td>£</td>
<td>Pound sign</td>
<td>£</td>
</tr>
<tr>
<td>¤</td>
<td>General currency sign</td>
<td>¤</td>
</tr>
<tr>
<td>¥</td>
<td>Yen sign</td>
<td>¥</td>
</tr>
<tr>
<td>†</td>
<td>Broken vertical bar</td>
<td>†</td>
</tr>
<tr>
<td>§</td>
<td>Section sign</td>
<td>§</td>
</tr>
<tr>
<td>–</td>
<td>Umlaut</td>
<td>–</td>
</tr>
</tbody>
</table>

Symbol Font Characters (see below)

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
<th>Code in MTD as</th>
</tr>
</thead>
<tbody>
<tr>
<td>Δ</td>
<td>Cap delta</td>
<td>#SYM##Δ##STD##</td>
</tr>
<tr>
<td>SYMBOL</td>
<td>DESCRIPTION</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>©</td>
<td>Copyright</td>
<td></td>
</tr>
<tr>
<td>*</td>
<td>Feminine ordinal</td>
<td></td>
</tr>
<tr>
<td>«</td>
<td>Left angle quote</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>Not sign</td>
<td></td>
</tr>
<tr>
<td>®</td>
<td>Registered trademark</td>
<td></td>
</tr>
<tr>
<td>–</td>
<td>Macron accent</td>
<td></td>
</tr>
<tr>
<td>°</td>
<td>Degree sign</td>
<td></td>
</tr>
<tr>
<td>±</td>
<td>Plus or minus</td>
<td></td>
</tr>
<tr>
<td>²</td>
<td>Superscript 2</td>
<td></td>
</tr>
<tr>
<td>³</td>
<td>Superscript 3</td>
<td></td>
</tr>
<tr>
<td>*</td>
<td>Acute accent</td>
<td></td>
</tr>
<tr>
<td>µ</td>
<td>Micro sign (Greek mu)</td>
<td></td>
</tr>
<tr>
<td>¶</td>
<td>Paragraph sign</td>
<td></td>
</tr>
<tr>
<td>†</td>
<td>Middle dot</td>
<td></td>
</tr>
<tr>
<td>́</td>
<td>Cedilla</td>
<td></td>
</tr>
<tr>
<td>́</td>
<td>Superscript 1</td>
<td></td>
</tr>
<tr>
<td>ो</td>
<td>Masculine ordinal</td>
<td></td>
</tr>
<tr>
<td>»</td>
<td>Right angle quote</td>
<td></td>
</tr>
<tr>
<td>¼</td>
<td>Fraction one-fourth</td>
<td></td>
</tr>
<tr>
<td>½</td>
<td>Fraction one-half</td>
<td></td>
</tr>
<tr>
<td>¾</td>
<td>Fraction three-fourths</td>
<td></td>
</tr>
<tr>
<td>ć</td>
<td>Inverted question mark</td>
<td></td>
</tr>
<tr>
<td>Ā</td>
<td>Capital A, grave accent</td>
<td></td>
</tr>
<tr>
<td>Ā</td>
<td>Capital A, acute accent</td>
<td></td>
</tr>
<tr>
<td>Ā</td>
<td>Capital A, circumflex accent</td>
<td></td>
</tr>
<tr>
<td>Ā</td>
<td>Capital A, tilde</td>
<td></td>
</tr>
<tr>
<td>Ā</td>
<td>Capital A, umlaut</td>
<td></td>
</tr>
<tr>
<td>Ā</td>
<td>Capital A, ring</td>
<td></td>
</tr>
<tr>
<td>ÅE</td>
<td>Capital AE ligature</td>
<td></td>
</tr>
<tr>
<td>Ç</td>
<td>Capital C, cedilla</td>
<td></td>
</tr>
<tr>
<td>Ė</td>
<td>Capital E, grave accent</td>
<td></td>
</tr>
<tr>
<td>Ė</td>
<td>Capital E, acute accent</td>
<td></td>
</tr>
<tr>
<td>Ė</td>
<td>Capital E,</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ê</td>
<td>Capital E, umlaut</td>
</tr>
<tr>
<td>Ï</td>
<td>Capital I, grave accent</td>
</tr>
<tr>
<td>Ï</td>
<td>Capital I, acute accent</td>
</tr>
<tr>
<td>Ï</td>
<td>Capital I, circumflex accent</td>
</tr>
<tr>
<td>Ï</td>
<td>Capital I, umlaut</td>
</tr>
<tr>
<td>Ð</td>
<td>Capital eth, Icelandic</td>
</tr>
<tr>
<td>Ñ</td>
<td>Capital N, tilde</td>
</tr>
<tr>
<td>Ò</td>
<td>Capital O, grave accent</td>
</tr>
<tr>
<td>Ò</td>
<td>Capital O, acute accent</td>
</tr>
<tr>
<td>Ò</td>
<td>Capital O, circumflex accent</td>
</tr>
<tr>
<td>Ò</td>
<td>Capital O, tilde</td>
</tr>
<tr>
<td>Ò</td>
<td>Capital O, umlaut</td>
</tr>
<tr>
<td>×</td>
<td>Multiply sign</td>
</tr>
<tr>
<td>Õ</td>
<td>Capital O, slash</td>
</tr>
<tr>
<td>Ü</td>
<td>Capital U, grave accent</td>
</tr>
<tr>
<td>Ü</td>
<td>Capital U, acute accent</td>
</tr>
<tr>
<td>Ü</td>
<td>Capital U, circumflex accent</td>
</tr>
<tr>
<td>Ü</td>
<td>Capital U, umlaut</td>
</tr>
<tr>
<td>Ý</td>
<td>Capital Y, acute accent</td>
</tr>
<tr>
<td>ß</td>
<td>Capital thorn, Icelandic</td>
</tr>
<tr>
<td>ß</td>
<td>Small sz ligature, German</td>
</tr>
<tr>
<td>å</td>
<td>Small a, grave accent</td>
</tr>
<tr>
<td>å</td>
<td>Small a, acute accent</td>
</tr>
<tr>
<td>å</td>
<td>Small a, circumflex accent</td>
</tr>
<tr>
<td>å</td>
<td>Small a, tilde</td>
</tr>
<tr>
<td>å</td>
<td>Small a, umlaut</td>
</tr>
<tr>
<td>å</td>
<td>Small a, ring</td>
</tr>
</tbody>
</table>
### Symbol Font Characters Note

Any character in the standard "Symbol" font may be used. However, it does not work to simply include the character using the Symbol font in Word. Instead, use "##SYM##" to switch to the symbol font. Then use the corresponding normal character then "##STD##" to switch back to the standard font. For example:

```
##SYM##D##STD##
```

Here the "D" character selects the "A" from the Symbol font.

You can see all the Symbol characters in the standard Windows "Character Map" program "Start/Programs/Accessories/Character Map". Switch back and forth between the Courier and Symbol fonts.
Prototype MTD File

The following is loosely based on part of the OMRON-US AMCD document.

Note that this file uses the MS Word Outline feature. When you view it in "Outline View", you can collapse and expand segments.

Document & Version

<table>
<thead>
<tr>
<th>Document</th>
<th>TestMTD00</th>
<th>This is a test MTD file.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1.0.0</td>
<td>7/25/99</td>
</tr>
<tr>
<td></td>
<td>1.0.1</td>
<td>7/26/99</td>
</tr>
<tr>
<td></td>
<td>JVR</td>
<td>Initial version</td>
</tr>
<tr>
<td></td>
<td>JVR</td>
<td>Minor tweaks</td>
</tr>
</tbody>
</table>

Java

<table>
<thead>
<tr>
<th>Java</th>
<th>Function isTuesday</th>
<th>boolean</th>
<th>return cal.get(Calendar.DAY_OF_WEEK)==Calendar.TUESDAY;</th>
<th>Define a Java variable</th>
</tr>
</thead>
</table>

Responses

<table>
<thead>
<tr>
<th>Command</th>
<th>Name</th>
<th>Options</th>
<th>Type</th>
<th>Validation</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>Label</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Personal Information</td>
</tr>
<tr>
<td>Response</td>
<td>FirstName</td>
<td>req</td>
<td>String</td>
<td>FirstNameValid == FirstName.length()&gt;0 &amp;&amp; FirstName.length()&lt;=25;</td>
<td>Enter your first name</td>
</tr>
<tr>
<td>Response</td>
<td>LastName</td>
<td>req</td>
<td>String</td>
<td>LastNameValid == LastName.length()&gt;0 &amp;&amp; LastName.length()&lt;=25;</td>
<td>Last name</td>
</tr>
<tr>
<td>Response</td>
<td>Age</td>
<td>int</td>
<td></td>
<td>AgeValid == (Age==1</td>
<td></td>
</tr>
<tr>
<td>Response</td>
<td>Gender</td>
<td>req</td>
<td>String</td>
<td>GenderValid == stringsEqual(Gender, &quot;male&quot;)</td>
<td></td>
</tr>
<tr>
<td>Response</td>
<td>IsSmoker</td>
<td>req</td>
<td>boolean</td>
<td></td>
<td>Are you a smoker?</td>
</tr>
<tr>
<td>Response</td>
<td>Weight</td>
<td>req</td>
<td>float</td>
<td>WeightValid == (Weight &gt;= -1)</td>
<td></td>
</tr>
<tr>
<td>Response</td>
<td>Height</td>
<td>req</td>
<td>float</td>
<td>HeightValid == (Height &gt;= -1)</td>
<td></td>
</tr>
<tr>
<td>Response</td>
<td>HBP</td>
<td>req</td>
<td>boolean</td>
<td></td>
<td>Do you have high blood pressure?</td>
</tr>
<tr>
<td>Response</td>
<td>IIA</td>
<td>req</td>
<td>boolean</td>
<td></td>
<td>Heart attack?</td>
</tr>
<tr>
<td>Response</td>
<td>ASM</td>
<td>req</td>
<td>boolean</td>
<td></td>
<td>Asthma?</td>
</tr>
<tr>
<td>Response</td>
<td>HiChol</td>
<td>req</td>
<td>boolean</td>
<td></td>
<td>Bi.</td>
</tr>
<tr>
<td>Response</td>
<td>CHF</td>
<td>req</td>
<td>boolean</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Variables and Parameters

<table>
<thead>
<tr>
<th>Command</th>
<th>Name</th>
<th>Options</th>
<th>Type</th>
<th>Code</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable</td>
<td>IsMale</td>
<td>boolean</td>
<td>IsMale</td>
<td><code>IsMale = stringsEqual(Gender, &quot;male&quot;);</code></td>
<td>Convenient variable</td>
</tr>
<tr>
<td>Variable</td>
<td>IsFemale</td>
<td>boolean</td>
<td>IsFemale</td>
<td><code>IsFemale = stringsEqual(Gender, &quot;female&quot;);</code></td>
<td>Ditto</td>
</tr>
<tr>
<td>Variable</td>
<td>GenderKnown</td>
<td>boolean</td>
<td>GenderKnown</td>
<td>`GenderKnown = IsMale</td>
<td></td>
</tr>
<tr>
<td>Variable</td>
<td>ManWoman</td>
<td>String</td>
<td>ManWoman</td>
<td><code>ManWoman = IsMale ? &quot;man&quot; : &quot;female&quot;;</code></td>
<td></td>
</tr>
<tr>
<td>Variable</td>
<td>YoungAge</td>
<td>int</td>
<td>YoungAge</td>
<td><code>YoungAge = 25;</code></td>
<td></td>
</tr>
<tr>
<td>Parameter</td>
<td>YoungMaleSmoker</td>
<td>boolean</td>
<td>YoungMaleSmoker</td>
<td>(Age&lt;YoungAge) &amp; &amp; IsMale &amp; &amp; IsSmoker;</td>
<td>This can be used below</td>
</tr>
<tr>
<td>Parameter</td>
<td>YoungFemaleSmoker</td>
<td>boolean</td>
<td>YoungFemaleSmoker</td>
<td>(Age&lt;YoungAge) &amp; &amp; IsFemale &amp; &amp; IsSmoker;</td>
<td>Saved in database</td>
</tr>
</tbody>
</table>
| Parameter     | BMI               | float       | BMI        | If (Weight>0, & & Height>0.) 
|               |                   |             | BMI = 704.5f * Weight / (Height * Height); | Check for known Weight & Height |
|               |                   |             |            | else 
|               |                   |             | BMI = -1; |                       |
| Parameter     | DiseaseCount      | int         | DiseaseCount | countOfBoolean (HBP, HA, ASM, H1Chol, CHF, ANG, OBD, DM, Stroke,TIA); | Number of reported diseases |
| Parameter     | BirthDateStr      | String      | BirthDateStr | dateToString(BirthDate); |                       |
| Variable      | DaysToQuit        | int         | DaysToQuit | `DaysToQuit = dateToday(QuitDate) - dateToday(today());` |                       |

### Introduction Section

<table>
<thead>
<tr>
<th>Command</th>
<th>Name</th>
<th>Options</th>
<th>Conditions</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group</td>
<td>Introduction</td>
<td></td>
<td></td>
<td>Your Personal Risk Reduction Program</td>
</tr>
<tr>
<td>Text</td>
<td></td>
<td></td>
<td></td>
<td>$$FirstName$, this &quot;booklet&quot; was designed using #H#computer technology#H# and your answers to the #HeartPoints# questionnaire. It is a unique health program based on the recommendations of medical experts and your needs and interests.</td>
</tr>
<tr>
<td>Group</td>
<td>HighBP</td>
<td>min</td>
<td>HBP</td>
<td>In particular, this program is intended to help you manage your hypertension and reduce your risk of developing cardiovascular disease (heart and blood vessels).</td>
</tr>
<tr>
<td>Text</td>
<td></td>
<td>max</td>
<td>HBP &amp; &amp; !ANG &amp; &amp; !CHF &amp; &amp; !Stroke</td>
<td>In particular, this program is intended to help you manage your hypertension and reduce your risk of developing cardiovascular disease (heart and blood vessels).</td>
</tr>
<tr>
<td>End Group</td>
<td>HighBP</td>
<td>NoHBP NoStroke</td>
<td>DiseaseCount</td>
<td>cardiovascular disease (heart and blood vessels).</td>
</tr>
<tr>
<td>-----------</td>
<td>--------</td>
<td>---------------</td>
<td>--------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>End Group</td>
<td>NoHBP</td>
<td>NoStroke</td>
<td>DiseaseCount</td>
<td>In particular, this program is intended to help you reduce your risk of developing hypertension and cardiovascular disease (heart and blood vessels).</td>
</tr>
<tr>
<td>Text</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Text</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>End Group</td>
<td>NoHBP</td>
<td>NoStroke</td>
<td>DiseaseCount</td>
<td>In particular, this program is intended to help you reduce your risk of developing hypertension and help you manage your cardiovascular disease (heart and blood vessels).</td>
</tr>
<tr>
<td>Text</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Text</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Group</td>
<td>YourHealthProfile</td>
<td></td>
<td></td>
<td>Your Health Profile</td>
</tr>
<tr>
<td>Text</td>
<td></td>
<td></td>
<td></td>
<td>In the questionnaire, you told us that you are a $$$Age$$$-year old $$$Man/Woman$$ with a history of the following</td>
</tr>
<tr>
<td>Text</td>
<td></td>
<td></td>
<td></td>
<td>Your personal history reveals the following</td>
</tr>
<tr>
<td>Text</td>
<td></td>
<td></td>
<td></td>
<td>condition that effects</td>
</tr>
<tr>
<td>Text</td>
<td></td>
<td></td>
<td></td>
<td>conditions that affect</td>
</tr>
<tr>
<td>Text</td>
<td></td>
<td></td>
<td></td>
<td>your cardiovascular system:</td>
</tr>
<tr>
<td>Text</td>
<td></td>
<td></td>
<td></td>
<td>• high blood pressure (hypertension)</td>
</tr>
<tr>
<td>Text</td>
<td></td>
<td></td>
<td></td>
<td>• heart attack</td>
</tr>
<tr>
<td>Text</td>
<td></td>
<td></td>
<td></td>
<td>• asthma</td>
</tr>
<tr>
<td>Text</td>
<td></td>
<td></td>
<td></td>
<td>• angina (chest pain)</td>
</tr>
<tr>
<td>Text</td>
<td></td>
<td></td>
<td></td>
<td>• congestive heart failure</td>
</tr>
<tr>
<td>Text</td>
<td></td>
<td></td>
<td></td>
<td>• high cholesterol (blood fats)</td>
</tr>
<tr>
<td>Text</td>
<td></td>
<td></td>
<td></td>
<td>• stroke</td>
</tr>
<tr>
<td>Text</td>
<td></td>
<td></td>
<td></td>
<td>• TIA</td>
</tr>
<tr>
<td>Text</td>
<td></td>
<td></td>
<td></td>
<td>• other heart disease</td>
</tr>
<tr>
<td>Text</td>
<td></td>
<td></td>
<td></td>
<td>You told us you do not have any medical conditions that increase your risk of cardiovascular (heart and blood vessel) disease.</td>
</tr>
<tr>
<td>Text</td>
<td></td>
<td></td>
<td></td>
<td>Your birthdate is $$$DateToString(BirthDate)$$$ and your quit date is is $$$DateToString(QuitDate)$$$</td>
</tr>
<tr>
<td>Text</td>
<td></td>
<td></td>
<td></td>
<td>It is $$$DaysToQuit$$$ days before your quit date.</td>
</tr>
<tr>
<td>Graphic</td>
<td>YoungManSmoking</td>
<td></td>
<td></td>
<td>It has been $$$DaysToQuit$$$ days since you quit.</td>
</tr>
<tr>
<td>Graphic</td>
<td>YoungWomanSmoking</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Graphic</td>
<td>YoungMaleSmoker</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Graphic</td>
<td>YoungFemaleSmoker</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Appendix B
package TJBuilder;

import java.io.*;
import java.util.*;
import HealthMedia.HMCore2.Utils.XMLUtils.*;
import java.text.MessageFormat;
import org.w3c.dom.*;
import com.ibm.xml.parsers.DOMParser;
import org.xml.sax.*;

/**
 * TJWriteJavaJob is a job class that writes tailored Java.
 * @author John Van Rooekel
 */
public class TJWriteJavaJob extends TJJob {

    protected static String lineSeparator = System.getProperty("line.separator");
    protected final int mOptionColumnNum = 2;  // Number of column containing options
    protected int mTextColumnNum = 4;         // Number of column containing tailoring text

    protected static final String fJavaSourceXMLFile = "TJEngineJava.xml";  // File containing Java source

    protected TJNode mDocNode = null;        // Top document node
    protected Node mJavaTextTopNode = null;  // Top XML node from Java source file
    protected int mNumResp = 0;              // Number of RESPONSE variables
    protected int mNumVars = 0;              // Number of VARIABLE variables
    protected int mNumParams = 0;            // Number of PARAMETER variables
    protected PrintWriter mWriter = null;   // Writer to output Java source to
    protected int mWriterLineNum = 0;       // Current line number in mWriter
    protected LinkedList mContentNames = new LinkedList();  // List of content names

    protected static final String subPrefix = "$$";  // Used to prefix Java substitutions
    protected static final String subSuffix = "$$";  // Used to suffix Java substitutions

    /*
     * Sets the text column to use.
     */
public void setTextColumn (int col) {
    mTextColumnNum = col;
}

/**
 * Write Java source based on HTML tree.
 * @param docNode TJNode containing the tree's document node
 * @param writer PrintWriter for the Java source output
 */
public void writeJava(TJNode docNode, PrintWriter writer) {
    if (docNode==null)
        throw new IllegalArgumentException("docNode is null");
    if (writer==null)
        throw new IllegalArgumentException("writer is null");

    try {
        mDocNode = docNode;
        mWriter = writer;
        mWriterLineNum = 1;

        // Read XML document containing Java source & keep top node.

        Document javaXMLDoc = HealthMedia.HMI.Core2.UTILS.XMLUtils.parseXML(new
                                BufferedReader(new FileReader("JavaSourceXMLFile")));
        mJavaTextTopNode = javaXMLDoc.getFirstChild();

        // Output the common preamble, including constant definitions and variables.

        String docName = mDocNode.getName();
        outputFormattedJavaText("Preamble1", new Object[][]{docName, mDocNode};
        writeConstants();
        outputFormattedJavaText("Preamble2", new Object[][]{docName, mDocNode};
        writeVariables();
        outputFormattedJavaText("Preamble3", new Object[][]{docName, mDocNode};

        // Output any included Java (pure pass-through from a java file).
        writeIncludedJava();

        // Output any user Java (JAVA and FUNCTION commands).

        writeGlobalUserJava();
        writeUserFunctions();

        // Output all the required methods.

        writegetResponseNames();
        writeGetInstrumentNames();
        writeValidateResponseValue();

    } catch (Exception e) { writer.println(e.getMessage()); }
writeGetOption();
writeCheckRequired();
writeGetVariableNames();
writeGetParameterNames();
writeComputeVariable();
writeSetVariableValue();
writeGetVariableValue();
writeGetVariableType();
writeGetCommand();
writeGetLabel();
writeGetObject();

// Output the "writeDocument" method.
writeWriteDocument();

// Now that we have filled the content name list, write the "getContentNames" method.
writeGetContentNames();

// Output the "writeVariableValues" method. Useful for debugging.
writeWriteVariableValues();

// Finally, output the final common Java code.
outputFormattedJavaText("Postamble", null, mDocNode);

} catch (Exception e) {
    handleException(e);
}

/**
 * Write the constants.
 *
 * public void writeConstants() throws Exception {
 *     for (Enumeration enum = mDocNode.getChildren(); enum.hasMoreElements(); ) {
 *         TJNode node = (TJNode)enum.nextElement();
 *         if (node.getType().equals("Response"))
 *             mNumResps++;
 *         if (node.getType().equals("Variable"))
 *             mNumVars++;
 *         if (node.getType().equals("Parameter"))
 *             mNumParams++;
 *     }
 *     outputFormattedJavaText("ConstantString", new Object[]{"fDocName", mDocNode.getName()})
 *     outputFormattedJavaText("Constant", new Object[]{"fNumResps", String.valueOf(mNumResps)})
 *     outputFormattedJavaText("Constant", new Object[]{"fNumVars", String.valueOf(mNumVars})
 * }
outputFormattedJavaText("Constant", new Object[] { "NumParams", String.valueOf(mNumParams) }, null);
}

/**
 * Write the variables.
 */
public void writeVariables() throws Exception {

    for (Enumeration enum = mDocNode.getChildren(); enum.hasMoreElements(); ) {
        TJNode node = (TJNode)enum.nextElement();
        if (node.isVariable()) {

            String name = node.getName();
            String type = getJavaType(node, 3);
            int index = type.indexOf(" ");
            if (index == 1) {
                String value = getDefaultValue(type);
                outputFormattedJavaText("Variable1", new Object[]{type, name, value, node});
            } else {
                String base = type.substring(0, index);
                String params = name + ", " + type.substring(index + 1);
                outputFormattedJavaText("Variable2", new Object[]{base, name, params}, node);
            }
        }
    }

    /**
     * Write user Java code - global.
     */
    public void writeGlobalUserJava() throws Exception {

        outputFormattedJavaText("UserJava1", null, null);
        for (Enumeration enum = mDocNode.getChildren(); enum.hasMoreElements(); ) {
            TJNode node = (TJNode)enum.nextElement();
            if (node.getType().equals("Java") &&
                node.checkOption(mOptionColumnNum, "global")) {
                String code = getJavaCode(node, 4);
                outputFormattedJavaText("UserJava2", new Object[]{code}, node);
            }
        }
    }

    /**
     * Write user Function code.
     */
    public void writeUserFunctions() throws Exception {

        for (Enumeration enum = mDocNode.getChildren(); enum.hasMoreElements(); ) {
            TJNode node = (TJNode)enum.nextElement();
            if (node.getType().equals("Function")) {

                String name = node.getTrimmedString(1);
                String type = getJavaType(node, 3);
                String code = getJavaCode(node, 4);
            }
        }
    }
public void writeIncludedJava() throws Exception {
    outputFormattedJavaText("IncludedJava1", null, null);
    for (Enumeration enum = mDOMNode.getChildren(); enum.hasMoreElements(); ) {
        TJNode node = (TJNode)enum.nextElement();
        if (node.getType().equals("IncludeJava")) {
            // retrieve the pathname of the file to include
            String filePath = TJRDFHTMLJob.getTrimmed(node.getName(), node.getRowText(),
            TJRDFHTMLJob.fNameColumnNum);
            File javaFile = new File(filePath);
            // verify that it exists and is readable
            if (javaFile.canRead()) {
                LineNumberReader reader = new LineNumberReader(new
            FileReader(javaFile));
                // read each line, writing it to our output file
                String code = reader.readLine();
                while (code != null) {
                    outputFormattedJavaText("IncludedJava2", new
                Object[] { code }, node);
                    code = reader.readLine();
                }
                // cleanup
                reader.close();
            } else {
                throw new IOException("Cannot find included java file: " +
            filePath);
            }
        }
    }
}

/*
 * Write writeResponseNames method.
 */
public void writeResponseNames() throws Exception {
    outputFormattedJavaText("GetResponseNames1", null, null);

    int i = 0;
    for (Enumeration enum = mDOMNode.getChildren(); enum.hasMoreElements(); ) {
        TJNode node = (TJNode)enum.nextElement();
        if (node.getType().equals("Response")) {
            String name = node.getName();
            outputFormattedJavaText("GetResponseNames2", new
```java
Object[][][String.valueOf(i++), name], node);
}

outputFormattedJavaText("GetResponseNames3", null, null);

/**
 * Write getContentNames method.
 */
public void writeGetContentNames() throws Exception {
    int count = mContentNames.size();
    outputFormattedJavaText("getContentNames1", new Object[][][String.valueOf(count)],
null);
    for (int i=0; i<count; i++) {
        String name = (String)mContentNames.get(i);
        outputFormattedJavaText("getContentNames2", new Object[][][String.valueOf(i), name], null);
    }

    outputFormattedJavaText("getContentNames3", null, null);
}

/**
 * Write writeGetInstrumentNames method.
 */
public void writeGetInstrumentNames() throws Exception {

    outputFormattedJavaText("getInstrumentNames1", null, null);
    int i = 0;
    for (Enumeration enum = mDocNode.getChildren(); enum.hasMoreElements(); ) {
        TJNode node = (TJNode)enum.nextElement();
        if (node.isInstrument()) {
            String name = node.getName();
            outputFormattedJavaText("getInstrumentNames2", new Object[][][String.valueOf(i++), name], node);
        }
    }

    outputFormattedJavaText("getInstrumentNames3", null, null);
}

/**
 * Write validateResponseValue method.
 */
public void writeValidateResponseValue() throws Exception {

    outputFormattedJavaText("validateResponseValue1", null, null);
    for (Enumeration enum = mDocNode.getChildren(); enum.hasMoreElements(); ) {
        TJNode node = (TJNode)enum.nextElement();
        if (node.getType().equals("Response")) {
```
String name = node.getName();
String code = getCode(node, 4);
if (code==null || code.length()==0)
    outputFormattedJavaText("ValidateResponseValue2",
        new Object[] {name, node});
else
    outputFormattedJavaText("ValidateResponseValue3",
        new Object[] {name, code, node});
}

outputFormattedJavaText("ValidateResponseValue4", null, null);

/**
 * Write getOption method.
 */
public void writeGetOption() throws Exception {
    outputFormattedJavaText("GetOption1", null, null);
    for (Enumeration enum = mDocNode.getChildren(); enum.hasMoreElements();)
    {
        TJNode node = (TJNode)enum.nextElement();
        if (node.isVariable() || node.isLabel())
        {
            String name = node.getName();
            String options = node.getTrimmedString(2);
            outputFormattedJavaText("GetOption2",
                new Object[] {name, options, node});
        }
    }
    outputFormattedJavaText("GetOption3", null, null);
}

/**
 * Write checkRequired method.
 */
public void writeCheckRequired() throws Exception {
    outputFormattedJavaText("CheckRequiredValue1", null, null);
    for (Enumeration enum = mDocNode.getChildren(); enum.hasMoreElements();)
    {
        TJNode node = (TJNode)enum.nextElement();
        if (node.getType().equals("Response") == true)
        {
            String name = node.getName();
            String options = node.getRowText(2);  
            String required = "false";
            if (options!=null)
            {
                int index = options.indexOf("req");
                if (index!=-1)
                    required = "true";
            }
            outputFormattedJavaText("CheckRequiredValue2",
                new Object[] {name, required, node});
        }
    }
}
outputFormattedJavaText("CheckRequiredValue3", null, null);

/**
 * Write getVariableNames method.
 */
public void writeGetVariableNames() throws Exception {

    outputFormattedJavaText("GetVariableNames1", null, null);

    int i = 0;
    for (Enumeration enum = mDocNode.getChildren(); enum.hasMoreElements(); ) {
        TJNode node = (TJNode)enum.nextElement();
        String type = node.getType();
        if (type.equals("Variable") || type.equals("Parameter")) {
            String name = node.getName();
            outputFormattedJavaText("GetVariableNames2",
                              new Object[]{String.valueOf(++i), name}, node);
        }
    }

    outputFormattedJavaText("GetVariableNames3", null, null);
}

/**
 * Write getParameterNames method.
 */
public void writeGetParameterNames() throws Exception {

    outputFormattedJavaText("GetParameterNames1", null, null);

    int i = 0;
    for (Enumeration enum = mDocNode.getChildren(); enum.hasMoreElements(); ) {
        TJNode node = (TJNode)enum.nextElement();
        String type = node.getType();
        if (type.equals("Parameter")) {
            String name = node.getName();
            outputFormattedJavaText("GetParameterNames2",
                              new Object[]{String.valueOf(++i), name}, node);
        }
    }

    outputFormattedJavaText("GetParameterNames3", null, null);
}

/**
 * Write computeVariable method.
 */
public void writeComputeVariable() throws Exception {

    outputFormattedJavaText("ComputeVariable1", null, null);

    int i = 0;
for (Enumeration enum = mDocNode.getChildren(); enum.hasMoreElements(); ) {
    TJNode node = (TJNode)enum.nextElement();
    String type = node.getType();
    if (type.equals("Variable") || type.equals("Parameter")) {
        String name = node.getName();
        String code = getJavaCode(node, 4);
        outputFormattedJavaText("ComputeVariable2",
                                new Object[]{name, code, node});
    }
}

outputFormattedJavaText("ComputeVariable3", null, null);

/**
 * Write setVariableValue method.
 */
public void writeSetVariableValue() throws Exception {
    outputFormattedJavaText("SetVariableValue1", null, null);
    for (Enumeration enum = mDocNode.getChildren(); enum.hasMoreElements(); ) {
        TJNode node = (TJNode)enum.nextElement();
        String type = node.getType();
        if (type.equals("Response") || type.equals("Parameter")) {
            String name = node.getName();
            String javaType = getJavaType(node, 3);
            if (javaType.equals("String"))
                outputFormattedJavaText("SetVariableValue2",
                                        new Object[]{name, node});
            else {
                int index = javaType.indexOf(" ");
                if (index != -1)
                    javaType = javaType.substring(0, index);  // Yes.
                javaType = javaType.substring(0, 1).toUpperCase();
                String converter = "+" + javaType;
                if (index == -1)  // Extended type?
                    outputFormattedJavaText("SetVariableValue3", null,
                                new Object[]{name, converter, node});
                else
                    outputFormattedJavaText("SetVariableValue4", null,
                                new Object[]{name, converter, node});
            }
        }
    }

    outputFormattedJavaText("SetVariableValue5", null, null);
}

/**
"* Write getVariableValue method."

public void writeGetVariableValue() throws Exception {
    outputFormattedJavaText("GetVariableValue1", null, null);
    for (Enumeration enum = mDocNode.getChildren(); enum.hasMoreElements(); ) {
        TJNode node = (TJNode)enum.nextElement();
        String type = node.getType();
        if (type.equals("Response") || type.equals("Variable") || type.equals("Parameter"))
        {
            String name = node.getName();
            String javaType = getJavaType(node, 3);
            String code = getConvertedVariable(javaType, name);
            if (isClassType(javaType) && javaType.equals("String"))
                outputFormattedJavaText("GetVariableValue2",
                    new Object[] { name, code, node });
            else
                outputFormattedJavaText("GetVariableValue3",
                    new Object[] { name, code, node });
        }
    }
    outputFormattedJavaText("GetVariableValue4", null, null);
}

"* Write getVariableType method."

public void writeGetVariableType() throws Exception {
    outputFormattedJavaText("GetVariableType1", null, null);
    for (Enumeration enum = mDocNode.getChildren(); enum.hasMoreElements(); ) {
        TJNode node = (TJNode)enum.nextElement();
        if (node.isVariable()) {
            String name = node.getName();
            String javaType = getJavaType(node, 3);
            outputFormattedJavaText("GetVariableType2",
                new Object[] { name, javaType, node });
        }
    }
    outputFormattedJavaText("GetVariableType3", null, null);
}

"* Write getCommand method."

public void writeGetCommand() throws Exception {
    outputFormattedJavaText("GetCommand1", null, null);
    for (Enumeration enum = mDocNode.getChildren(); enum.hasMoreElements(); ) {
        TJNode node = (TJNode)enum.nextElement();
    }
}
if (node.isVariable() || node.isLabel()) {
    String name = node.getName();
    String command = node.getTrimmedString(0);
    outputFormattedJavaText("GetCommand2",
        new Object[]{name,command}, node);
}
outputFormattedJavaText("GetCommand3", null, null);

/**
 * Write getLabel method.
 */
public void writeGetLabel() throws Exception {
    outputFormattedJavaText("GetLabel1", null, null);
    for (Enumeration enum = mDocNode.getChildren(); enum.hasMoreElements(); ) {
        TJNode node = (TJNode)enum.nextElement();
        if (node.isVariable() || node.isLabel()) {
            String name = node.getName();
            String label = node.getTrimmedString(5);
            if (label==null || label.length()==0)
                label = name;
            label = replaceAll(label, "\"", "\"");
            outputFormattedJavaText("GetLabel2",
                new Object[]{name,label}, node);
        }
    }
outputFormattedJavaText("GetLabel3", null, null);
}

/**
 * Write getObject method.
 */
public void writeGetObject() throws Exception {
    outputFormattedJavaText("GetObject1", null, null);
    for (Enumeration enum = mDocNode.getChildren(); enum.hasMoreElements(); ) {
        TJNode node = (TJNode)enum.nextElement();
        if (node.isVariable()) {
            String objectString = null;
            String name = node.getName();
            String type = getJavaType(node, 3);
            if (isClassType(type)) {
                objectString = name;
            } else if (type.equals("boolean")) {
                objectString = "new Boolean(" + name + ");
            } else if (type.equals("float")) {
                objectString = "new Float(" + name + ");
            } else if (type.equals("double")) {
                objectString = "new Double(" + name + ");
            } else if (type.equals("char")) {
                objectString = name;
            }
        }
    }
}
objectString = "new Character(" + name + ")";
}
else if (type.equals("int")) {
    objectString = "new Integer(" + name + ")";
}
else if (type.equals("short")) {
    objectString = "new Short(" + name + ")";
}
else if (type.equals("long")) {
    objectString = "new Long(" + name + ")";
}
else if (type.equals("byte")) {
    objectString = "new Byte(" + name + ")";
}
else {
    objectString = "null";
}
outputFormattedJavaText("GetObject2",
    new Object[]{name,objectString}, node);
}
outputFormattedJavaText("GetObject3", null, null);

/**
 * Write writeDocument method.
 */
public void writeWriteDocument() throws Exception {
    String docName = mDocNode.getName();
    outputFormattedJavaText("WriteDocument1", new Object[]{docName}, mDocNode);
    writeSelectionCalls(mDocNode, 1);
    outputFormattedJavaText("WriteDocument2", null, mDocNode);
    writeGroupMethods(mDocNode, 1);
}

/**
 * Write a series of calls to selection methods.
 */
public void writeSelectionCalls(TJNode groupNode,
    int level) throws Exception {
    String groupName = groupNode.getName();

    for (Enumeration enum = groupNode.getChildren(); enum.hasMoreElements(); ) {
        TJNode node = (TJNode) enum.nextElement();
        String type = node.getType();
        if (type.equals("Text") || type.equals("Graphic") || type.equals("Group") ||
            type.equals("Content") || type.equals("Table") ||
            type.equals("Chart")
            || (type.equals("Java") && 
            node.checkOption(mOptionColumnNum, "global"))) {
            String name = node.getName();
        }
String condition = getJavaCode(node, 3);
if (condition.endsWith(":"))
    throw new Exception("Condition ends with ";)
else
    String call = null;
    String skip = null;
    /*
     * Text
     */
    if (type.equals("Text")) {
        String text = getTailoringText(node, mTextColumnNum);
        //call = getFormattedJavaText("WriteTextCall",
        //new Object[]\{name, text\});
        call = getWriteTextCall(name, text);
    }
    /*
     * Graphic
     */
    } else if (type.equals("Graphic")) {
        call = getFormattedJavaText("WriteGraphicCall",
                                   new Object[]\{name\});
    }
    /*
     * Group
     */
    } else if (type.equals("Group")) {
        String methodName = "write" + name;
        call = getFormattedJavaText("WriteGroupCall",
                                   new Object[]\{methodName\});
    }
    /*
     * Content
     */
    } else if (type.equals("Content")) {
        String code = getJavaCode(node, 4);
        call = getFormattedJavaText("WriteContentCall",
                                   new Object[]\{code\});
    }
    /*
     * Table
     */
    } else if (type.equals("Table")) {
        String data = getData(node, 4);
        call = getFormattedJavaText("WriteTableCall",
                                   new Object[]\{name, data\});
    }
    /*
     * Chart
     */
    } else if (type.equals("Chart")) {
        String data = getData(node, 4);
        call = getFormattedJavaText("WriteChartCall",
                                   new Object[]\{name, data\});
    }
    /*
     * Java
     */
    } else if (type.equals("Java")) {
        String code = getJavaCode(node, 4);
        call = getFormattedJavaText("UserJava3",
                                   new Object[]\{code\});


private String getWriteTextCall(String name, String text) throws Exception {
    // break the text into separate chunks when we write it into the Java file -
    // this is purely cosmetic, and helps with debugging in JBuilder, which only supports
    // lines of about 800 characters in the editor
    String chunkedText = getChunkedText(text);
    String call = getFormattedJavaText("WriteTextCall", new Object[] {name, chunkedText});
    return call;
}

private String getChunkedText(String text) {
    if (text.length() < 850)
        return text;
    StringBuffer chunkedText = new StringBuffer();
    int currentLineLength = 0;
    boolean inQuote = true;
    char c;
    for (int i=0; i<text.length(); i++) {
        c = text.charAt(i);
        if (c == "") {
            if (inQuote == false)
                inQuote = true;
            else
                chunkedText.append(c);
        } else {
            chunkedText.append(c);
            currentLineLength++;
        }
    }
    return chunkedText.toString();
}
```java
else
    inquote = false;
    chunkedText.append(c);
}
else if (c == ' ') {
    if (inquote && (currentline.length > 750)) {
        chunkedText.append(c);
        chunkedText.append('"');
        chunkedText.append('+');
        chunkedText.append('n');
        chunkedText.append('"');
        currentline.length = 0;
    }
    else
        chunkedText.append(c);
}
else
    chunkedText.append(c);
    currentline.length++;
}
return chunkedText.toString();
}

/**
 * Write a series of group methods. May recurse.
 */
public void writeGroupMethods(TreeNode groupNode,
   int level) throws Exception {

    String groupName = groupNode.getName();

    for (Enumeration enum = groupNode.getChildren(); enum.hasMoreElements(); ) {
        TreeNode node = (TreeNode)enum.nextElement();
        String type = node.getType();
        if (type.equals("Group")) {
            String name = node.getName();
            String methodName = "write" + name;
            outputFormatedJavaText("WriteGroupMethod1",
                new Object[] {name, methodName, String.valueOf(level)},
                node);
            writeSelectionCalls(node, level);
            outputFormatedJavaText("WriteGroupMethod2",
                new Object[] {name, String.valueOf(level), node});

            writeGroupMethods(node, level+1);
        }
    }
}

/**
 * Write the writeVariableValues method
 */
public void writeWriteVariableValues() throws Exception {
outputFormattedJavaText("WriteVariableValues1", null, null);

for (Enumeration enum = mDocNode.getChildren(); enum.hasMoreElements(); ) {
    TTreeNode node = (TTNode)enum.nextElement();
    String variableType = node.getType();
    if (node.isVariable()) {
        String javaType = getJavaType(node, 3);
        String name = node.getName();
        String code = getConvertedVariable(javaType, name);
        outputFormattedJavaText("WriteVariableValues2", new 
        Object[]{variableType, javaType, name, code}, node);
    }
}

outputFormattedJavaText("WriteVariableValues3", null, null);

/**
 * Get text from node column, trimmed.
 *
 * @param node TTreeNode containing node.
 * @param column int containing column.
 * @return String with text.
 */
protected String getColumnText(TTreeNode node, int column) throws Exception {
    String text = node.getTrimmedString(column);
    return text;
}

/**
 * Get Java type from node column.
 *
 * @param node TTreeNode containing node.
 * @param column int containing column.
 * @return String with formatted type.
 */
protected String getJavaType(TTreeNode node, int column) throws Exception {
    String type = getColumnText(node, column);
    if (type.indexOf("\n") == -1)
        throw new Exception("Java type contains a line break \"+type+\"\n");
    return type;
}

/**
 * Get Java code from node column.
 *
 * @param node TTreeNode containing node.
 * @param column int containing column.
 * @return String with formatted code.
 */
protected String getJavaCode(TTreeNode node, int column) throws Exception {
    String code = getColumnText(node, column);
}
protected String fixJavaCode(String code) throws Exception {
  code = fixJavaCode(code);
  return code;
}

protected String fixJavaCode(String code) throws Exception {
  code = fixJavaCode(code);
  code = replaceAll(code, "##Idq##", "\"");
  code = replaceAll(code, "##rdq##", "\"");
  code = replaceAll(code, "##BoldOn##", "\"");
  code = replaceAll(code, "##BoldOff##", "\"");
  code = replaceAll(code, "##quot##", "\"");
  return code;
}

/**
 * Get tailoring text from node column.
 */
* @param node TJNode containing node.
* @param column int containing column.
* @return String with formatted text.
*/
protected String getTailoringText(TJNode node, int column) throws Exception {
  String text = node.getRowText(column);
  text = replaceAll(text, "\n", "\n");
  text = replaceAll(text, "\n", "\t\n");
  text = replaceAll(text, "u2022", "\n\t\n");
  text = replaceAll(text, "\n", "\n");
  return text;
}

/**
 * Get data from node column (for Tables and Charts)
 */
* @param node TJNode containing node.
* @param column int containing column.
* @return String with data.
*/
protected String getData(TJNode node, int column) throws Exception {
  String data = getColumnText(node, column);
  data = replaceAll(data, "##par##", "\n");
  data = replaceAll(data, "##\n\n##", "\n");
  data = replaceAll(data, "##"\n\n##", "\n");
  return data;
}

/**
 * Format a string to insert Java calls.
 */
* @param text String to format.
*/
protected String formatJava(String text) { return text; }
protected String substituteJavaCalls(String text) throws Exception {
    StringBuffer buf = new StringBuffer();
    for (int i=0; i<text.length(); i) {
        int j = text.indexOf(subPrefix, i);
        if (j===-1) {
            buf.append(text.substring(i));
            break;
        }
        int k = j + subPrefix.length();
        int l = text.indexOf(subSuffix, k);
        if (l===-1) {
            throw new Exception("Closing \"\+subSuffix\+\" missing from \"\+text\+\";\n            buf.append(text.substring(i, l));
            buf.append("\+\"\+\")
            // now put the real quotes (\") back in the javacall
            String javacall = text.substring(k, l);
        }
    }
    return buf.toString();
}

/**
 * Get as Java code containing converted string of a variable.
 */

protected String getConvertedVariable(String javaType, String name) {
    if (javaType.equals("String"))
        return name;
    else {
        StringBuffer converter = new StringBuffer();
        converter.append("toString();
        converter.append(name);
        converter.append("\"\";
        return converter.toString();
    }

    /* Get a named piece of Java text. */

protected String getJavaText (String name) throws Exception {
    String str =
}
HealthMedia.HMICore2.UTILs.XMLUtils.getNodeValueByName(mJavatxtTopNode, name, "TEXT");
    if (str==null)
        throw new Exception("Cannot find Java text for name "+name);
    str = fixLineSeparators(str);
    return str;
}

/**Get formatted Java text.*/
protected String getFormattedJavaText(String name, Object[] args) throws Exception {
    String pattern = getJavaText(name);
    String text = null;
    try {
        text = MessageFormat.format(pattern, args);
    } catch (IllegalFormatException e) {
        throw new IllegalFormatException(e.toString()+", "+pattern+"\n");
    }
    return text;
}

/**Output Java text with argument substitution.*/
protected void outputFormattedJavaText(String name, Object[] args, TjNode node) throws Exception {
    String output = getFormattedJavaText(name, args);
    mWriter.print(output);
    int start = mWriterLineNum;
    for (int index=output.indexOf(lineSeparator,0); index!=-1;
            index=output.indexOf(lineSeparator,index)) {
        mWriterLineNum++;
        index++;
    }
    if (node==null &
            (node.getProperty().equals("IncludeJava")))
        node.addLineNumbers(start, mWriterLineNum);
}

/**Replace all occurrences of one string with another.*/
protected String replaceAll(String str, String from, String to) {
    StringBuffer buf = new StringBuffer();
    for (int start=0; ) {
        int index = str.indexOf(from, start); // Search for "from" string
        if (index==-1) {
            buf.append(str.substring(start)); // No more occurrences, copy the rest
            break;
        }
        buf.append(str.substring(index, start) + to); // Append text up to "from" string
        start = index + from.length(); // Append "to" string
    }
    return buf.toString();
}
protected String fixLineSeparators(String str) {
    if (str.indexOf(lineSeparator)!=-1) // Already has line separators
        return str;
    return replaceAll(str, "\n", lineSeparator);
}

/**Determine whether type is numeric.*/
protected boolean isNumericType(String type) {
    return type.equals("byte") || type.equals("short") || type.equals("int") || type.equals("long")
        || type.equals("char") ||
        type.equals("float") || type.equals("double");
}

/**Determine whether type is a class.*/
protected boolean isClassType(String type) {
    return !isNumericType(type) && type.equals("boolean");
}

/**Given a type, return a default value.*/
protected String defaultValue(String type) {
    if (isNumericType(type))
        return "-1";
    else if (type.equals("boolean"))
        return "false";
    else
        return "null";
}
CLAIMS

What is claimed is:

1. A method for generating computer source code, comprising:
   preparing a master tailoring document which contains logic for a computer program;
   inputting the master tailoring document into a document structure generator, wherein the document structure generator parses out the computer program logic contained in the input document and builds a document tree structure based on the program logic; and
   providing a code writer program and a template of pre-defined source code snippets, wherein the code writer program retrieves the computer program logic from the document tree structure and combines the retrieved program logic with designated source code snippets from the template to generate source code for a computer program.

2. The source code generation method of claim 1, further comprising compiling the generated source code to convert the generated source code into executable code.

3. The source code generation method of claim 2, further comprising:
   providing computer memory and a computer workstation;
   moving the executable code to the computer memory;
   activating the executable code, which generates a dynamic user interface on the computer workstation;
   inputting data into the dynamic user interface and submitting the data;
   processing the input data; and
   creating an output content document.

4. The source code generation method of claim 3, further comprising:
   formatting the output content document into a formatted file; and
   transmitting the formatted file to the computer workstation for review.
5. The source code generation method of claim 4, wherein formatting the output content document into a formatted file includes:
   providing a hypertext markup language (HTML) publisher and an HTML layout file; and
   inputting the output content document and the HTML layout file into the HTML publisher to format the formatted file in HTML.

6. The source code generation method of claim 4, wherein formatting the output content document into a formatted file includes:
   providing a print publisher and a print layout file; and
   inputting the output content document and the print layout file into the print publisher to format the formatted file in a print file format.

7. The source code generation method of claim 4, wherein formatting the output content document into a formatted file includes formatting the output content document into a text file format.

8. The source code generation method of claim 1, wherein:
   the master tailoring document includes commands that set document tree structure nodes when parsed by the document structure generator.

9. The source code generation method of claim 8, wherein:
   the master tailoring document includes commands that create branches from each generated node when parsed by the document structure generator.

10. The source code generation method of claim 8, wherein:
    the master tailoring document includes group commands having subcommands that create branches from each generated node when parsed by the document structure generator.

11. The source code generation method of claim 1, wherein the template of pre-defined source code snippets is written in extensible markup language (XML).
12. The source code generation method of claim 1, wherein the generated source code is written in the Java programming language.

13. The source code generation method of claim 1, wherein the master tailoring document includes at least one condition statement written to output text content based on the condition statement.

14. The source code generation method of claim 1, wherein the master tailoring document includes at least one condition statement written to output graphics content based on the condition statement.

15. A system for generating computer source code from a master tailoring document which contains logic for a computer program, comprising:

   a code generator having:
   a document structure generator;
   a memory;
   a code writer; and
   a template of pre-defined source code snippets, wherein the document structure generator parses out the computer program logic contained in the master tailoring document and builds a document tree structure in memory based on the program logic and wherein the code writer program retrieves the computer program logic from the document tree structure and combines the retrieved program logic with designated source code snippets from the template to generate source code for a computer program.

16. The source code generation system of claim 15, further comprising a code compiler for compiling the generated source code to convert the generated source code into executable code.
17. The source code generation system of claim 15, wherein:
the master tailoring document includes commands that set document tree
structure nodes when parsed by the document structure generator.

18. The source code generation system of claim 17, wherein:
the master tailoring document includes commands that create branches
from each generated node when parsed by the document structure generator.

19. The source code generation system of claim 17, wherein:
the master tailoring document includes group commands having
subcommands that create branches from each generated node when parsed by the
document structure generator.

20. The source code generation system of claim 15, wherein the template of
pre-defined source code snippets is written in extensible markup language (XML).

21. The source code generation system of claim 15, wherein the generated
source code is written in the Java programming language.

22. The source code generation method of claim 1, wherein the master tailoring
document includes at least one condition statement written to output text content based on
the condition statement.

23. The source code generation method of claim 1, wherein the master tailoring
document includes at least one condition statement written to output graphics content
based on the condition statement.
<xml version="1.0"?>
<JTEngineJava>

<!--==Preamble1, (0)=Doc Name ==-------------------------------==-->  
<TEXT Name="Preamble1"><![CDATA[

/*
 * {0} Tailored Java Engine
 *
 * Copyright 1999 by HealthMedia, Inc.
 *
 */

import java.util.*;
import java.io.*;
import java.text.*;

import TJEngine.TJEngineBase;
import TJEngine.TJMTDEngine.*;

public class {0} extends TJEngineBase {
    * Misc constants
    */
}
]]></TEXT>

<!--==Preamble2, (0)=Doc Name ==-------------------------------==-->  
<TEXT NAME="Preamble2"><![CDATA[

    public {0} {
        
        /*
         * RESPONSE, VARIABLE and PARAMETER variables
        */
    }
]]></TEXT>

<!--==Constant, (0)=Constant Name, (1)=Constant Value ==------==-->  
<TEXT NA protected static final int {0} = " {1};
]]></TEXT>

<!--==ConstantString, (0)=Constant Name, (1)=Constant Value ==---==-->  
<TEXT NA protected static final String {0} = " {1}" ;
]]></TEXT>

<!--==Preamble3 ==-----------------------------------------------==-->  
<TEXT NAME="Preamble3"><![CDATA[

    /* fMIN and fMAX variables

SUBSTITUTE SHEET (RULE 26)
```java
protected boolean fMin = false;
protected boolean fMax = false;
```

```
// Included Java

```xml
<TEXT NAME=" IncludedJava1"><![CDATA[
  *
  * Included Java
  */
]]>
</TEXT>

```xml
<TEXT NAME=" IncludedJava2, {0}=Java code "><![CDATA[
  *
  * User Java and User Functions
  */
]]>
</TEXT>

```xml
<TEXT NAME=" IncludedJava3, {0}=Java code "><![CDATA[
  ' (0)'
]]>
</TEXT>

```xml
<!-- User Function, (0)=Function name, (1)=Type (2)=Args, (3)=Code-->
<TEXT NAME=" UserFunction"> <![CDATA[
  *
  * (0)
  */
  @return
  */
  public {1} {0}[(2)] {3}
}]]>
</TEXT>

```xml
<TEXT NAME=" Variable1, {0}=Variable Type, {1}=Variable Name, {2}=Variable Value-->
<TEXT NA "protected {0} {1} = {2};" ]]>]]>
</TEXT>

```xml
<TEXT NAME=" Variable2, {0}=Variable Type, {1}=Variable Name, {2}=CSV params "><![CDATA[
]]>
</TEXT>
```

SUBSTITUTE SHEET (RULE 26)
<TEXT NA protected {0} {1} = new {0}("{2}");
]]><</TEXT>

]]>------ Postamble =------------------------------------------------------------------>
<TEXT NAME="Postamble"><![CDATA[
]]>------ GetResponseNames1 =------------------------------------------------------------------>
<TEXT NAME="GetResponseNames1"><![CDATA[
/**
* Get all RESPONSE names
* @return Vector of Strings containing a list of all RESPONSE variable names
* /
public Vector getResponseNames () {'
    Vector respNames = new Vector(fNumResp);
    return respNames;
}
]]>------ GetResponseNames2, {0}=Index, {1}=Response Name =------------------------------------------------------------------>
<TEXT NAME="GetR respNames.add(0, '{1}');
]]>------ GetResponseNames3 =------------------------------------------------------------------>
<TEXT NAME="GetResponseNames3"><![CDATA[
    return respNames;
}
]]>------ GetContentNames1, {0}=count =------------------------------------------------------------------>
<TEXT NAME="GetContentNames1"><![CDATA[
/**
* Get all CONTENT names
* @return Vector of Strings containing list of all CONTENT variable names
* /
public Vector getContentNames () {'
    Vector contentNames = new Vector(0);
    return contentNames;
}
]]>------ GetContentNames2, {0}=Index, {1}=Content Name =------------------------------------------------------------------>

```
<TEXT NAME="GetContentNames3">GetContentNames3 = <CDATA[
    return contentNames;
}
]]><TEXT>

[...]

public Vector getInstrumentNames() {
    Vector instNames = new Vector();
    [...]

public boolean validateResponseValue(String name) throws NoSuchFieldException {
    [...]

SUBSTITUTE SHEET (RULE 26)
9/27

FIG. 5E

```
<!----- ValidateResponseValue2, (0)=Response Name =========>
<Text NAME="Valida if (name.equalsIgnoreCase("(0)")) '{
  return true;
}' 340

<!----- ValidateResponseValue3, (0)=Response name, (1)=Validation Code=====>
<Text NAME="Valida if (name.equalsIgnoreCase("(0)")) '{
  boolsen (0)Valid = true;
  (1)
  return (0)Valid;
}'

<!----- ValidateResponseValue4 ===============>
<Text NAME="Valida throw new NoSuchFieldException("Unknown response name, "+name);">

<!----- GetOption1==============>
<Text NAME="GetOption1"><![CDATA[
  /**
   * Get the Option text for a variable.
   *
   * @param name String containing the name of the variable
   * @return String containing the option text
   */
   public String getOption(String name) throws NoSuchFieldException '{

<!----- GetOption2, (0)=Response Name, (1)=Option Text (no quotes) =========>
<Text NAME="GetF if (name.equalsIgnoreCase("(0)")) return (1);

<!----- GetOption3 ===============>
<Text NAME="GetF throw new NoSuchFieldException("Unknown variable name, "+name);

<!----- GetCommand1==============>
```

SUBSTITUTE SHEET (RULE 26)
FIG. 5F

```java
public String getCommand(String name) throws NoSuchFieldException {
    /* Get the Command text for a variable or label. */
    @param name String containing the name of the variable
    @return String containing the command text
*/
    if (name.equalsIgnoreCase("0"))
        return "{1}";
}
```

```java
public String getCommand2, {0}=Variable Name, {1}=Command Text (no quotes) ========-->
if (name.equalsIgnoreCase("0"))
    return "{1}";
```

```java
GetC throw new NoSuchFieldException("Unknown variable name, +name);
```

```java
public String getLabel(String name) throws NoSuchFieldException {
    /* Get the label text for a variable or label. */
    @param name String containing the name of the variable
    @return String containing the label text
*/
    if (name.equalsIgnoreCase("0"))
        return "{1}";
}
```

```java
GetLA if (name.equalsIgnoreCase("0"))
    return "{1}";
```

```java
GetLabel3 throw new NoSuchFieldException(" Unknown variable name, +name);
```

```java
GetObject1 GetObject1>"GetObject1">"GetObject1";
```

* Get an Object representing the value of a variable.
* @param name String containing the name of the variable
* @return Object representing the value of the variable
*/
public Object getObject(String name) throws NoSuchFieldException {
    
426  <!-- GetObject2, {0}=Variable Name, {1}=Label Text (no quotes) -->
  <TEXT NAME="GetO if (name.equalsIgnoreCase("{0}"))
             return {1};
  
428  <!-- GetObject3 -->
  <TEXT NAME=" GetO throw NoSuchFieldException(" Unknown variable name, " +name);
  
360  <!-- CheckRequiredValue1 -->
  <TEXT NAME=" CheckRequiredValue1"><![CDATA[
     /**
      * Check whether a response is required.
      *
      * @param name String containing the name of the response
      * @return Boolean, true if the response is required
      */
    public boolean checkRequired (String name) throws NoSuchFieldException {

374  <!-- GetVariableNames1 -->
  <TEXT NAME=" GetVariableNames1"><![CDATA[
     /**
      * Get a list of all VARIABLE/PARAMETER names
      *
      * @return Vector of strings containing VARIABLE/PARAMETER names
      */
FIG. 5H

```
public Vector getVariableNames () {'
    Vector names = new Vector(NumVars+NumParams);
}

<------------------------ GetVariableNames2, (0)=Index, (1)=Variable Name ------------------------>

<TEXT NAME="" GetV names.add((0), " (1) ");
}

<------------------------ GetVariableNames3 -------------------------------------------------------->

<TEXT NAME="" GetVariableNames3"> <![CDATA[
    return names;
}]

<------------------------ GetParameterNames1 ------------------------------------------------------>

<TEXT NAME="" GetParameterNames1"> <![CDATA[
    */
    * Get a list of all just PARAMETER names
    */
    * @return Vector of Strings containing PARAMETER names
    */
    public Vector getParameterNames () {'
        Vector names = new Vector(NumParams);
    }
}

<------------------------ GetParameterNames2, (0)=Index, (1)=Variable Name ------------------------>

<TEXT NAME="" GetP names.add((0), " (1) ");
}

<------------------------ GetParameterNames3 -------------------------------------------------------->

<TEXT NAME="" GetParameterNames3"> <![CDATA[
    return names;
}]

<------------------------ ComputeVariable1 --------------------------------------------------------->

<TEXT NAME="" ComputeVariable1"> <![CDATA[
    */
    * Compute the value of single VARIABLE or PARAMETER variable using
    * the characterization snippet from the MTD
    */
    * @param name String containing the name of the PARAMETER or VARIABLE
    */
    public void computeVariable (String name) throws Exception {'
}
```
FIG. 5J

 GetVariableValue1

 ```java
/**
 * Get the value of a single RESPONSE/VARIABLE/PARAMETER variable
 * @param name String containing the name of the variable
 * @return value String containing the value
 */
public String getVariableValue (String name) throws NoSuchFieldException, ParseException {

```

 GetVariableValue2

 ```java
/**
 * GetV if (name.equalsIgnoreCase("{0}")) ' {
 if ((0)==null)
 return null;
 else
 return {1};

```

 GetVariableValue3

 ```java
/**
 * GetV if name.equalsIgnoreCase("{0}")) ' {
 return {1};

```

 GetVariableValue4

 ```java
/**
 * GetV throw new NoSuchFieldException(" Unknown variable name," +name);

```

 GetVariableType1

 ```java
/**
 * Get the Java type of a single RESPONSE/VARIABLE/PARAMETER variable
 * @param name String containing the name of the variable
 * @return type String containing the Java type
 */
public String getVariableType (String name) throws NoSuchFieldException {

```

 GetVariableType2

 ```java
/**
 * GetV if (name.equalsIgnoreCase("{0}")) ' {
 return " {1}" ;

```
FIG. 5K

```xml
<TEXT NAME="GetVariableType3"/>

<TEXT NAME="GetV throw new NoSuchFieldException(\" Unknown variable name, \" +name);"/>
```

```xml
<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>

<TEXT NAME="WriteDocument1"/>
FIG. 5M

```xml
<TEXT NAME="WriteVariableValues1">
  /**
   * Write the names and values of all variables
   */
  public void writeVariableValues() throws Exception {
}
</TEXT>

<!-- WriteVariableValues2, {0}=Variable Type, {1}=Java Type, {2}=Name,
    {3}=Code=-->
<TEXT NAME="Write write Variable(" {0}" , " {1}" , " {2}" , " {3}" ;
</TEXT>

<!-- WriteVariableValues3=---------------------------------------->
<TEXT NAME="WriteVariableValues3"></TEXT>
</TEXT>
</JTEngineJava>
/*
 * SimpleTest Tailored Java Engine
 * Copyright 1999 by HealthMedia, Inc.
 * This software is the confidential and proprietary information of HealthMedia, Inc.
 */

import java.util.*;
import java.io.*;
import java.text.*;
import TJEngine.TJEngineBase;
import TJEngine.TJMTDEngine.*;

public class SimpleTest extends TJEngineBase {
    /*
    * Misc constants
    */
    protected static final String fDocName = "SimpleTest";
    protected static final int fNumResp = 3;
    protected static final int fNumVars = 0;
    protected static final int fNumParams = 0;

    public SimpleTest () {
    }

    /*
    * RESPONSE, VARIABLE and PARAMETER variables
    */
    protected String FirstName = null;
    protected String LastName = null;
    protected int Age = -1;

    /*
    * fMin and fMax variables
    */
    protected boolean fMin = false;
    protected boolean fMax = false;

    /*
    * Included Java
    */

    /*
    * User Java and User Functions
    */

    /**
    * Get all RESPONSE names
    *
    * @return Vector of Strings containing a list of all RESPONSE variable names
    */

    SUBSTITUTE SHEET (RULE 26)
public Vector getResponseNames () {
    Vector respNames = new Vector (iParamResp);
    respNames.add(0, "FirstName");
    respNames.add(1, "LastName");
    respNames.add(2, "Age");
    return respNames;
}

/**
 * Get all INSTRUMENT names - include RESPONSE, LABEL, and ENDLABEL types
 * @return Vector of Strings containing a list of all
 *     RESPONSE, LABEL, or ENDLABEL variable names
 */
public Vector getInstrumentNames () {
    Vector instNames = new Vector ();
    instNames.add(0, "FirstName");
    instNames.add(1, "LastName");
    instNames.add(2, "Age");
    return instNames;
}

/**
 * Validate a single response using the validation snippet from the MTD
 * @param name String containing the name of the response
 * @return Boolean, true if valid, false otherwise
 */
public boolean validateResponseValue (String name) throws NoSuchFieldException
{
    if (name.equalsIgnoreCase ("FirstName")) {
        return true;
    }
    if (name.equalsIgnoreCase ("LastName")) {
        return true;
    }
    if (name.equalsIgnoreCase ("Age")) {
        return true;
    }
    throw new NoSuchFieldException ("Unknown response name, " + name);
}

/**
 * Get the Option text for a variable.
 * @param name String containing the name of the variable
 * @return String containing the option text
 */
public String getOption (String name) throws NoSuchFieldException {
    
SUBSTITUTE SHEET (RULE 26)
if (name.equalsIgnoreCase("FirstName"))
    return "354
if (name.equalsIgnoreCase("LastName"))
    return "356
if (name.equalsIgnoreCase("Age"))
    return "358
    throw new NoSuchFieldException("Unknown variable name, \* +name");
}

/**
 * Check whether a response is required.
 *
 * @param name String containing the name of the response
 * @return Boolean, true if the response is required
 */
public boolean checkRequired (String name) throws NoSuchFieldException {
    if (name.equalsIgnoreCase("FirstName"))
        return false; 368
    if (name.equalsIgnoreCase("LastName"))
        return false; 370
    if (name.equalsIgnoreCase("Age"))
        return false; 372
    throw new NoSuchFieldException("Unknown response name, \* +name");
}

/**
 * Get a list of all VARIABLE/PARAMETER names
 *
 * @return Vector of Strings containing VARIABLE/PARAMETER names
 */
public Vector getVariableNames () {
    Vector names = new Vector (fNumVars+fNumParams); 380
    return names;
}

/**
 * Get a list of all just PARAMETER names
 *
 * @return Vector of Strings containing PARAMETER names
 */
public Vector getParameterNames () {
    Vector names = new Vector (fNumParams); 388
    return names;
}

/**
 * Compute the value of single VARIABLE or PARAMETER variable using
 * the characterization snippet from the MTD
 */

SUBSTITUTE SHEET (RULE 26)
```java
392 */
393 public void computeVariable (String name) throws Exception {
394     throw new Exception(" Unknown variable name, " + name);
395 }
396 */
397 */
398 * Set the value of a single RESPONSE/VARIABLE/PARAMETER variable
399 */
400 * @param name String containing the name of the variable
401 * @param value String containing the value
402 *
403 public void setVariableValue (String name, String value) throws
404     NoSuchFieldException, ParseException {
405     if (name.equalsIgnoreCase(" FirstName" )) {
406         FirstName = value;
407         return;
408     }
409     if (name.equalsIgnoreCase(" LastName" )) {
410         LastName = value;
411         return;
412     }
413     if (name.equalsIgnoreCase(" Age" )) {
414         Age = toInt(value);
415         return;
416     }
417     throw new NoSuchFieldException(" Unknown variable name, " + name);
418 }
419 */
420 */
421 * Get the value of a single RESPONSE/VARIABLE/PARAMETER variable
422 */
423 * @param name String containing the name of the variable
424 * @return value String containing the value
425 *
426 public String getVariableValue (String name) throws
427     NoSuchFieldException, ParseException {
428     if (name.equalsIgnoreCase(" FirstName" )) {
429         return FirstName;
430     }
431     if (name.equalsIgnoreCase(" LastName" )) {
432         return LastName;
433     }
434     if (name.equalsIgnoreCase(" Age" )) {
435         return toString(Age);
436     }
437     throw new NoSuchFieldException(" Unknown variable name, " + name);
438 }
439 */
440 */
441 * Get the Java type of a single RESPONSE/VARIABLE/PARAMETER variable
442 */
443 * @param name String containing the name of the variable
444 */
```
@return type  String containing the Java type

```java
public String getVariableType (String name) throws NoSuchFieldException {
    if (name.equalsIgnoreCase("FirstName")) {
        return "String" ;
    }
    if (name.equalsIgnoreCase("LastName")) {
        return "String" ;
    }
    if (name.equalsIgnoreCase("Age")) {
        return "int" ;
    }
    throw new NoSuchFieldException(" Unknown variable name, " +name);
}
```

/**
 * Get the Command text for a variable or label.
 */

```java
@param name  String containing the name of the variable
@return  String containing the command text
```

```java
public String getCommand (String name) throws NoSuchFieldException {
    if (name.equalsIgnoreCase("FirstName"))
        return "Response" ;
    if (name.equalsIgnoreCase("LastName"))
        return "Response" ;
    if (name.equalsIgnoreCase("Age"))
        return "Response" ;
    throw new NoSuchFieldException(" Unknown variable name, " +name);
}
```

/**
 * Get the Label text for a variable or label.
 */

```java
@param name  String containing the name of the variable
@return  String containing the label text
```

```java
public String getLabel (String name) throws NoSuchFieldException {
    if (name.equalsIgnoreCase("FirstName"))
        return "First Name: " ;
    if (name.equalsIgnoreCase("LastName"))
        return "Last Name: " ;
    if (name.equalsIgnoreCase("Age"))
        return "Age: " ;
    throw new NoSuchFieldException(" Unknown variable name, " +name);
}
```

/**
 * Get an Object representing the value of a variable.
 */

```java
@param name  String containing the name of the variable
@return  Object representing the value of the variable
```

SUBSTITUTE SHEET (RULE 26)
public Object getObject (String name) throws NoSuchFieldException {
    if (name.equalsIgnoreCase(" FirstName" ))
        return FirstName;
    if (name.equalsIgnoreCase(" LastName" ))
        return LastName;
    if (name.equalsIgnoreCase(" Age" ))
        return new Integer (Age);
    throw new NoSuchFieldException(" Unknown variable name, " + name);
}

/**
 * Write the tailored document content
 * @param output PrintStream for output
 */
public void writeDocument (PrintWriter writer, boolean min, boolean max) throws Exception {
    fMin = min;
    fMax = max;
    mWriter = writer;
    mLevel = 0;
    writeDocumentHeader(" SimpleTest");
    writeSimpleTestWelcome();
    writeVariableValues();
    writeDocumentTrailer();
}

/**
 * Write the SimpleTestWelcome group content
 * @param output PrintStream for output
 */
public void writeSimpleTestWelcome () throws Exception {
    writeGroupHeader(" SimpleTestWelcome");
    writeText(" SimpleTestWelcomeText0", " Welcome " +(FirstName)++") ;
    if (Age<=21)
        writeText(" SimpleTestWelcomeText1", " You are under 21. ");
    if (Age>=21)
        writeText(" SimpleTestWelcomeText2", " You are 21 or older. ");
    writeGroupTrailer(" SimpleTestWelcome");
}

/**
 * Get all CONTENT names
 */
public Vector getContentnames () {

SUBSTITUTE SHEET (RULE 26)
24/27

Vector contentNames = new Vector(4);
contentNames.add(0, "SimpleTestWelcome");
contentNames.add(1, "SimpleTestWelcomeText0");
contentNames.add(2, "SimpleTestWelcomeText1");
contentNames.add(3, "SimpleTestWelcomeText2");

return contentNames;

/**
 * Write the names and values of all variables
 */
public void writeVariableValues() throws Exception {
    writeVariable("Response", "String", "FirstName", FirstName);
    writeVariable("Response", "String", "LastName", LastName);
    writeVariable("Response", "int", "Age",toString(Age));
}

SUBSTITUTE SHEET (RULE 26)
FIG. 8

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<DOCUMENT NAME="SimpleTest" FORMAT="2.0">
  <GROUP NAME="SimpleTestWelcome" Level="1">
    <TEXT NAME="SimpleTestWelcomeText0" LEVEL="2">Welcome John,
    <TEXT NAME="SimpleTestWelcomeText1" LEVEL="2">You are 21 or
    <TEXT NAME="SimpleTestWelcomeText2" LEVEL="2">or older.]]>
      <GROUP>
        <VARIABLE VARIABLETYPE="Response" JAVATYPE="String" NAME="FirstName" VALUE="John" />
        <VARIABLE VARIABLETYPE="Response" JAVATYPE="String" NAME="LastName" VALUE="Doe" />
        <VARIABLE VARIABLETYPE="Response" JAVATYPE="int" NAME="Age" VALUE="22" />
      </GROUP>
  </GROUP>
</DOCUMENT>
```
Welcome John. You are 21 or older.
INTERNATIONAL SEARCH REPORT

International application No.
PCT/US01/10860

A. CLASSIFICATION OF SUBJECT MATTER
IPC(7) : G06F 17/21
US CL. : 707/513, 530, 526, 517
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED
Minimum documentation searched (classification system followed by classification symbols)
U.S. : 707/513, 530, 526, 517

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

<table>
<thead>
<tr>
<th>Category*</th>
<th>Citation of document, with indication, where appropriate, of the relevant passages</th>
<th>Relevant to claim No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>US 5,895,476 A (ORR et al) 20 April 1999, all.</td>
<td>1-23</td>
</tr>
<tr>
<td>A</td>
<td>US 6,023,714 A (HILL et al) 08 February 2000, all.</td>
<td>1-23</td>
</tr>
<tr>
<td>A</td>
<td>US 6,012,098 A (BAYEH et al) 04 January 2000, all.</td>
<td>1-23</td>
</tr>
<tr>
<td>A</td>
<td>US 5,860,073 A (FERREL et al) 12 January 1999, all.</td>
<td>1-23</td>
</tr>
</tbody>
</table>

[X] Further documents are listed in the continuation of Box C. [ ] See patent family annex.

* Special categories of cited documents:
  *A* document defining the general state of the art which is not considered to be of particular relevance
  *E* earlier document published on or after the international filing date
  *L* document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
  *O* document referring to an oral disclosure, use, exhibition or other means
  *P* document published prior to the international filing date but later than the priority date claimed
  *T* later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
  *X* document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
  *Y* document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
  *S* document member of the same patent family

Date of the actual completion of the international search: 06 AUGUST 2001

Date of mailing of the international search report: 24 AUG 2001

Name and mailing address of the ISA/US Commissioner of Patents and Trademarks
Box PCT
Washington, D.C. 20231
Facsimile No. (703) 305-3230

Authorized officer
STEPHEN HONG
Telephone No. (703) 305-3900

Form PCT/ISA/210 (second sheet) (July 1998) *
<table>
<thead>
<tr>
<th>Category*</th>
<th>Citation of document, with indication, where appropriate, of the relevant passages</th>
<th>Relevant to claim No.</th>
</tr>
</thead>
</table>