webSurvent/ webCATI

VERSION 8.1

A TECHNICAL USER'S MANUAL

COMPUTERS for MARKETING CORPORATION

547 Howard Street ♦ San Francisco, CA 94105 ♦ U.S. A.

TABLE OF CONTENTS

CHAPTER 1

OVERVIEW

What are webSurvent and webCATI?

CfMC’s webSurvent and webCATI are two products that work with a browser over the Internet or Intranet. The difference between them is that webSurvent is designed for respondents in the public who access the study at their leisure, while webCATI is designed for trained interviewers using telephones to call respondents under some form of sample and quota control, who then enter the results into the computer.

In practical terms, this means webSurvent has more stringent display requirements because respondents are the public, so more attention must be paid to the “look and feel” of the pages. Each respondent acts independently and either completes the survey or not.

WebCATI surveys are generated using telephone sample provided by the client. The surveys are scheduled using the CfMC phone sample management system, and executed by interviewers talking to respondents by phone. Webcati requires management of the sample and quotas as target populations are called and fill the survey population requirements. A supervisor is generally used to manage the interviewers, sample, and quotas.

To summarize, webSurvent and webCATI are products that:

Allow for Computer-Assisted Telephone Interviewing (CATI) through the Internet and/or an Intranet.

Have a browserinterviewing interface, which means a point-and-click capability for interviewers and a graphical display for respondents.

Include the full functionality of CfMC’s CATI package, Survent, including flexible quotas, complex logic conditions, and suspending/resuming interviews.

Utilize study files (Data, Sample, Quota) that are identical to Survent.

In addition, webCATI:

Permits multiple call centers to work on the same projects as long as they each have an internet/intranet connection.

Makes it possible for call-center interviewing agents to work from home via the internet/intranet.

Allows studies to be run using telnet sessions in conjunction with browser-based interactions (mixed CATI and webCATI).

How webSurvent works

WebSurvent scripts are written using the same command language as Survent with the addition of HTML and JavaScript code for enhanced screen displays.

Study files are located on a machine that hosts a Web server (e.g., Apache) and a CfMC study server.

Respondents access the study using a computer that can access the Internet/Intranet using Internet Explorer (IE), Firefox or similar browser.

Respondents are e-mailed an invitation or sent to the site via a link on the internet

Respondents access the URL and log into the survey

How webCATI works

WebCATI scripts are written using the same command language as Survent with the addition of HTML and JavaScript code for enhanced screen displays.

Interviewing files are located on the machine that hosts the Web server (e.g., Apache) and a CfMC study server.

Interviewers access the study using a computer that can access the Internet/Intranet using Internet Explorer (IE), Firefox or a similar browser.

Interviewers work from an interviewing site/phone room, an offsite location, or their home.

Interviewers access the URL and log into the survey.

The two programs use the same script language and interface. In addition, all files used and created are the same format as Survent.

For purposes of this manual, "CfMC's Web software" will be used when discussing both products. Each will be mentioned by name when referring to specific features of webSurvent or webCATI.

Support requirements

A basic understanding of Survent and its networking programs, including the study server and the Supervisor.

The ability to setup and maintain a UNIX computer and an Internet / Intranet connection.

Minimal proficiency in HTML and JavaScript.

Suggested reading

"Kevin Werbach's Bare Bones Guide to HTML" at http://www.werbach.com/barebones for a list of HTML commands.

A good HTML reference book is the O'Reilly guide: "*HTML: The Definitive Guide*" by Chuck Musciano, Bill Kennedy, Mike Loukides (editor) O'Reilly & Associates; ISBN: 1565924924

Recommended Web design skills

At least a basic understanding of JavaScript (We recommend another O'Reilly book: "*JavaScript: The Definitive Guide,"* by David Flanagan, O'Reilly & Associates; ISBN: 1565923928).

Knowledge of basic graphic design.

The following links give a good introduction to HTML:

http://www.cwru.edu/help/introHTML/;

The index lives at http://www.cwru.edu/help/introHTML/AppA.html

http://archive.ncsa.uiuc.edu/General/Internet/WWW/HTMLPrimerAll.html

http://www.utoronto.ca/webdocs/HTMLdocs/NewHTML/htmlindex.html

CfMC provides supplementary basic HTML training if necessary. Basic HTML and JavaScript knowledge need to be framed within the context of (Web) Survent programming.

Visit the CfMC demo site

CfMC has a demo that provides many working examples of programming. Go to:

http://demo.cfmc.com/exam81

CHAPTER 2

••••••••••••••••••••

GETTING STARTED

This chapter describes how to put up a questionnaire on the internet. Detailed instructions for each step are also provided later in this document. *See Appendix I* for a description of the system installation needed before using CfMC's Web software.

Scripting tools

In order to run a CFMC survey, you must create a Survent questionnaire script. This can be done by using CfMC's QuickSurvent script creation program, or by using the editor of your choice to create the commands needed for the script. If using an editor, it is important to use an ASCII text editor that does not format the page with special characters, because CfMC will not interpret the characters properly.

Many users find it easiest to use a WINDOWS or MAC text editor such as TEXTPAD or WORDPAD to create the scripts, and then save the file to the UNIX computer to be compiled and tested with SURVENT, but you can edit the file on the UNIX computer as well. There are many text editors available, contact support@cfmc.com if you need some suggestions.

Creating a work area for the study

You should have a specific area on the computer to keep the files separate for each study. You can make a separate directory to work in in your login area, or have a standard "studies" directory where you put the files for each study. Here, you will compile the questionnaire script, create the sample control files, report on the data collected, generate other files and reports, etc.

For webSurvent users, we provide a specific area you may use for your studies ($CFMC/websurv/studies) so you can make your studies there and we will point there by default for study files when you load the study.

Once the survey files are compiled and ready to use, you can use the "loadstudy" command to load the files under the CfMC server for interviewing.

Questionnaire script requirements:

In order to load a questionnaire script for internet display, the following options must be specified in the questionnaire header statement.

SCREENLINES=1000	Specify this to allow Web pages to be as long as you want them to be. By default, Survent expects to allow 23 or 24 lines per screen for terminal mode.
MAX_QUESTION_SIZE=64000	Specify this because the additional HTML code used by CfMC Web programs often will require larger question sizes.

You will also need to specify the data locations to use in the survey. It is suggested that you use the same locations for all surveys as follows:

CASE_LENGTH=32000	This is the maximum case length.
WORK_START=18001	This is where data that you define as "working" data goes, such as calculations or rotation seeds you don't want to pass on to the client, or data added after the questionnaire was started.
TEXT_START=20001	This is where the "open end" question data is stored.

You will need to add standard HTML enhancements in the body of the questionnaire (See more about Adding HTML enhancements in Chapter 4):

{HTML_RADIOBUTTONS=#}	This specifies when to use radio buttons and when to use a drop-down box, if you always want radio buttons, set this to "1". If you want a drop down box if there are more than 10 responses, set this to 10.
{HTML_CHECKBOXES=#}	This is the same as above, but for "multiple response" questions which use check boxes instead of radio buttons.
{!WEB_SURVENT}	This guarantees that you will not use features that won't work in the Web version of Survent.

It is best to use controls to specify standard HTML for different parts of a question. You can control the standard display of different parts of the question by using {!HTML_<keyword>} commands, for instance:

{!HTML_TEXT_PREFIX=<BR>}	To add a line break before the text of the question.

This reduces the amount of code you need to write and establishes the "look and feel" for the survey. See more about Adding HTML enhancements in Chapter 5 below.

Loading a study

Compile the questionnaire spec file by entering:

mentor <spec file> -<list file> ←

The "spec file" is a file that has the commands to create the questionnaire; the "list file" is a file where the program writes any comments or error messages about the compilation. If you get error messages, you need to find the errors, fix them, and compile again until there are no errors.

Create a phone file for the study:

For webSurvent, create a phone file of names and passwords with the MAKEFONE utility.

Type:

makefone <studycode> ← (for open studies)

makefone <studycode> <ASCII sample file> ← (closed studies)

For webCATI, create a phone sample file using the FONEBULD utility (see the sample management chapter; note that webCATI does NOT use passwords).

Type:

fonebuld

shopfile (to specificy shop parameters)

go

<study>

Sample.asc (ascii sample file)

Quit

WebCATI uses phone sample files to store phone numbers, names and any other information associated with a pool of potential respondents for a study.

NOTE: *See the SURVENT Technical Manual, Chapter 6, Section 1,* for more information on building phone and phone index files with the FONEBULD utility.

Move compiled study files (.qff, .quo,.fon,.fnx) into the CfMC server's study directories by using the LOADSTUDY utility.

Type:

loadstudy <studycode> ←

Create the index file (starting page) and other HTML files for the study and move files into your Web area with the WEBUTIL (webSurvent) or WCATUTIL (webCATI) utility.

Enter necessary variables when prompted (i.e., study code, open or closed).

Enter formatting options for HTML files when prompted (i.e., background color, image location, etc.)

Test the questionnaire by activating a Web browser (e.g. Netscape) and typing the Web site address and path to the questionnaire on the "go to" line:

www.<Web address>/<complete path to questionnaire>

For webSurvent, enter a name and password to start. For webCATI, enter an Interviewer ID on the index.html page. (Valid interviewer IDs are stored in the Employee Information file employee.xxx (in ${CFMC}ipcfiles/). Test the questionnaire on your browser to make sure it's running properly.

For webSurvent, make sure you test multiple browsers since you don't know which browser respondents will be using. For webCATI, you can just test using the browser the interviewers will be using.

For webSurvent, send e-mail invitations to prospective respondents, you can use CfMC's GUI-interfaced utility program e-Mailer, if desired. With this utility you provide names and passwords for closed studies. Instructions for how to use CfMC's e-Mailer application are included in a separate document. Contact doc@cfmc.com for that document.

For webCATI, make sure you load the proper sample and have proper quota referencess specified before starting.

CHAPTER 3

ADDING HTML ENHANCEMENTS

Formatting and programming techniques

Although CfMC Survent specifications will run on webSurvent without adding any HTML code, adding HTML enhancements produces a much more interesting and aesthetically pleasing questionnaire that might just entice respondents to take part in it. One thing you will want to do is combine questions on one screen using grids. Another is to set up a standard “look and feel” for all questions.

Typically, a webCATI-moderated interview requires less HTML page formatting than a webSurvent self-administered questionnaire. Some pages in a webCATI interview, such as an initial call disposition screen, may be more elaborate. (See the standard shell (wcshell.qpx) for an example call disposition screen.)

Using grids

Grids are used in CfMC’s Web programs any time you want to put more than one question on the screen at the same time. To specify a grid, type the {!GRID} command followed by your questions, then {!END_GRID}. By default, grids require that all questions be answered. You may specify that particular questions may be unanswered by using the question subype “B” on that question, or you may specify !GRID,B if you wish to allow all questions to be left blank or unanswered.

Grids in CfMC’s Web programs may have a !ROTATE/!ENDROTATE block inside a !GRID to rotate lists of attributes to be rated or other items. Also see the !FIELD question subtype R to rotate codes on a particular question.

Grids show any answers previously responded to, even when backing up to the grid. Without a grid, some questions will not keep their answers when you back up to them. Therefore, some questionnaires have every question that requires a response contained in a !GRID.

You will often build HTML “tables” for formatting inside of grids. The tables are generally used to group similar questions so that they are formatted in columns and rows. The HTML syntax starts with “<table>” and ends with “</table>”. See the examples for some typical table formats.

When you make Web pages in !GRIDs, keep in mind how much information you want to display per page; if you display too much information, the respondent will have to do a lot of scrolling. If you don’t put enough questions on each page, the respondent will have to click on the “next” button more often.

Adding standardized HTML code to control page structure

The following commands will automatically add HTML code to specified places in questions and response lists for uniform formatting; the commands are either HTML_<keyword>_PREFIX or HTML_<keyword>_SUFFIX commands. These have been combined in the documentation below. The shorthand version of the command follows the longhand version of some of these commands.

{!HTML_ERROR_PREFIX/SUFFIX= } {!HTML_ERR_PRE/SUF= }

Adds the defined HTML code to the beginning/end of error messages generated by webSurvent; in this way you can make the error messages a different size and color, or format them to fit nicely inside HTML tables.

{!HTML_GRIDERROR_PREFIX/SUFFIX= } {!HTML_GRIDERR_PRE/SUF= }

This is similar to the above code, but it affects the error messages that appear at the top of the grid. If there is an error on a grid question, an error message is displayed at the top of the page as well as on the specific question within the grid.

NOTE: This can only be set outside of a grid. This is because this command affects the entire page and not just a part of the page Therefore it cannot be changed while inside a grid.

{!HTML_LABELTAG}

This ADA-compliant feature will provide a label on each response item to tie the response text to the input tag. As a result, when you "mouse over" text, it is highlighted, and you can "click" on the text in order to mark the response as chosen. This is a default feature. To turn it off, add {!-HTML_LABELTAG} in the questionnaire specs. You may have to turn this feature off for some complex page controls or for some browsers, as well.

If you want to make the default such that this feature is NOT used by default, specify ">dump W7" in the file $CFMC/control/mentinit. The {!HTML_LABELTAG} command will still override the default in this case.

{!HTML_PAGE_PREFIX/SUFFIX= } {!HTML_PG_PRE/SUF= }

Adds defined HTML code to the beginning/end of the (whole) page.

NOTE: This can only be set outside of a grid.

{!HTML_QUESTION_TEXT_PREFIX/SUFFIX= } {!HTML_TEXT_PRE/SUF= }

Text_prefix would place the assigned code before the text of a question. Text_suffix would place it after the text of the question.

{!HTML_RESPONSE_TABLE_PREFIX/SUFFIX= } {!HTML_RT_PRE/SUF= }

Adds defined HTML code to the beginning or end of a response list.

{!HTML_RESPONSE_LIST_ITEM_PREFIX/SUFFIX= } {!HTML_RL_PRE/SUF= }

Adds defined HTML code to the beginning or end of a particular response list item, that is, before the response code, and after the response text for all items.

{!HTML_RESPONSE_LIST_COMMENT_PREFIX/SUFFIX= } {!HTML_RLC_PRE/SUF= }

Adds defined HTML code to the beginning orend of comment-type response list items (==code type items). These are items that have text but do not have a checkbox or radio button.

{!HTML_RESPONSE_CODE_PREFIX/SUFFIX= } {!HTML_RC_PRE/SUF= }

Adds defined HTML code to the beginning or end of the response code if you are displaying response codes on the list.

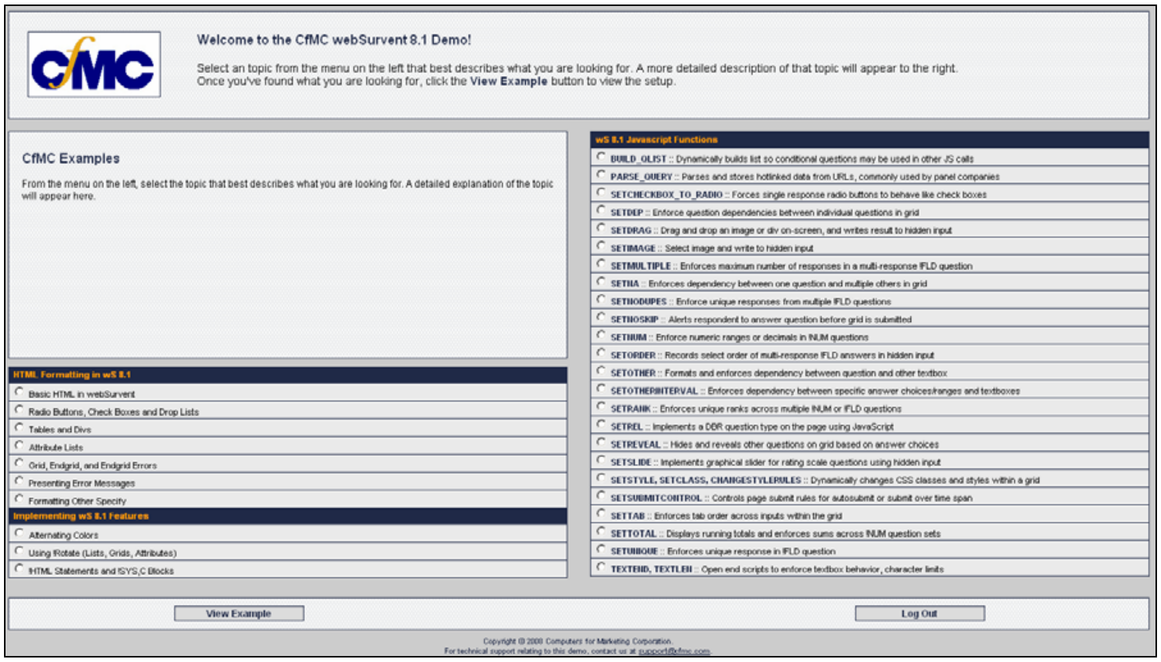
{!HTML_RESPONSE_TEXT_PREFIX/SUFFIX= } {!HTML_RT_PRE/SUF= }

Adds defined HTML code to the beginning or end of the response text of all items on the response list. That is, after the response code but before the response text, or directly after the response text of all items on the list.

{!HTML_NUMBER_EXCEPTION_TEXT= } {!HTML_NUM _TEXT= }

Adds text or defined HTML code between a numeric entry box and any exception codes. This is on numeric type questions only.

Go to http://demo.cfmc.com/exam81 for numerous examples in creating Web survey screens using HTML and CfMC programming techniques. It looks like this:



Welcome to the CfMC webSurvent 8.1 Demo!

Select an topic from the menu on the left that best describes what you are looking for. A more detailed description of that topic will appear to the right.
Once you've found what you are looking for, click the **View Example** button to view the setup.

CfMC Examples

From the menu on the left, select the topic that best describes what you are looking for. A detailed explanation of the topic will appear here.

HTML Formatting in wS 8.1

- Basic HTML in webSurvent
- Radio Buttons, Check Boxes and Drop Lists
- Tables and Divs
- Attribute Lists
- Grid, Endgrid, and Endgrid Errors
- Presenting Error Messages
- Formatting Other Specify

Implementing wS 8.1 Features

- Alternating Colors
- Using !Rotate (Lists, Grids, Attributes)
- !HTML Statements and !SYS,C Blocks

wS 8.1 Javascript Functions

- **BUILD_OLIST** :: Dynamically builds list so conditional questions may be used in other JS calls
- **PARSE_QUERY** :: Parses and stores hotlinked data from URLs, commonly used by panel companies
- **SETCHECKBOX_TO_RADIO** :: Forces single response radio buttons to behave like check boxes
- **SETDEP** :: Enforce question dependencies between individual questions in grid
- **SETDRAG** :: Drag and drop an image or div on-screen, and writes result to hidden input
- **SETIMAGE** :: Select image and write to hidden input
- **SETMULTIPLE** :: Enforces maximum number of responses in a multi-response FLD question
- **SETNA** :: Enforces dependency between one question and multiple others in grid
- **SETNODUPES** :: Enforce unique responses from multiple FLD questions
- **SETNOSKIP** :: Alerts respondent to answer question before grid is submitted
- **SETNUM** :: Enforce numeric ranges or decimals in NUM questions
- **SETORDER** :: Records select order of multi-response FLD answers in hidden input
- **SETOTHER** :: Formats and enforces dependency between question and other textbox
- **SETOTHERINTERVAL** :: Enforces dependency between specific answer choices/ranges and textboxes
- **SETRANK** :: Enforces unique ranks across multiple NUM or FLD questions
- **SETREL** :: Implements a DBR question type on the page using JavaScript
- **SETREVEAL** :: Hides and reveals other questions on grid based on answer choices
- **SETSLIDE** :: Implements graphical slider for rating scale questions using hidden input
- **SETSTYLE, SETCLASS, CHANGESTYLERULES** :: Dynamically changes CSS classes and styles within a grid
- **SETSUBMITCONTROL** :: Controls page submit rules for autosubmit or submit over time span
- **SETTAB** :: Enforces tab order across inputs within the grid
- **SETTOTAL** :: Displays running totals and enforces sums across NUM question sets
- **SETUNIQUE** :: Enforces unique response in FLD question
- **TEXTEND, TEXTLEN** :: Open end scripts to enforce textbox behavior, character limits

Copyright © 2008 Computers for Marketing Corporation.
For technical support relating to this demo, contact us at support@cfmc.com.

Formatting features

Defining, calling blocks of HTML commands using the !SYS,C command

There are multiple !HTML commands that you can use to give your surveys a certain look. These blocks of commands can be used in conjunction with the sys,c command that can be used when you want to have the same look and feel from screen to screen on your Web surveys.

You can define a block of these commands and call them anywhere in your spec where you want to use them.

To define a block:

Begin with a label for the block and the command !start_cd_block.

{default: !start_cd_block}

Place all !HTML commands to be used in this block. It is recommended that you define all of the !HTML commands for a particular cd block to insure that all commands are set exactly as you intend them to be. Then, end this block of commands with:

{!end_cd_block}

When you want to use this block of code in your spec, simply place a sys,c command that references the name of the block.

{!sys,c, default}

This block of code will apply until another !sys,c command is called. You can change individual commands at any time. Keep in mind that once changes are made to individual commands they will apply until another block is called or the individual command is changed again.

Here is an example of what you can do with the commands mentioned above.

```
{!HTML_DATAQUESTIONS_ONLY }


{DEFAULT: !start_cd_block  }
{!html_error_prefix=<span class='error'>}
{!html_error_suffix=</span><br/><br/>}
{!html_question_text_prefix=<table class='qtable'><tr><td colspan='2'
class='qtxt'> }
{!html_question_text_suffix=<br/><br/></td></tr> }
{!html_response_code_prefix=<tr><td class='radio'> }
{!html_response_code_suffix=</td> }
{!html_response_text_prefix=<td class='qlabel'> }
{!html_response_text_suffix=</td></tr> }
{!html_response_list_item_prefix= }
{!html_response_list_item_suffix= }
{!html_response_list_comment_prefix=<tr><td class='comment'>}
```

```
{!html_response_list_comment_suffix=</td></tr>}
{!html_response_table_prefix= }
{!html_response_table_suffix=</table><br/> }
{!html_num_input_prefix=<tr><td class='num' colspan='2'> }
{!html_num_input_suffix=</td></tr></table><br/> }
{!html_textquestion_input_prefix=<tr><td class='textq' colspan='2'> }
{!html_textquestion_input_suffix=</td></tr></table><br/> }
{!html_var_input_prefix=<tr><td class='var' colspan='2'> }
{!html_var_input_suffix=</td></tr></table><br/>  }
{!html_num_exception_prefix=<span class='numexp'> }
{!html_num_exception_suffix=</span> }
{!end_cd_block  }

{RATING: !start_cd_block  }
{!html_error_prefix=<tr class='error'><td colspan='6'>}
{!html_error_suffix=</td></tr>}
{!html_question_text_prefix=<tr><td> }
{!html_question_text_suffix=</td> }
{!html_response_code_prefix= }
{!html_response_code_suffix= }
{!html_response_text_prefix= }
{!html_response_text_suffix= }
{!html_response_list_item_prefix= <td align='center'> }
{!html_response_list_item_suffix= </td> }
{!html_response_list_comment_prefix= }
{!html_response_list_comment_suffix= }
{!html_response_table_prefix= }
{!html_response_table_suffix= }
{!html_num_input_prefix= }
{!html_num_input_suffix= }
{!html_textquestion_input_prefix= }
{!html_textquestion_input_suffix= }
{!html_var_input_prefix= }
{!html_var_input_suffix= }
{!html_num_exception_prefix= }
{!html_num_exception_suffix= }
{!end_cd_block  }
```

```
{!SYS,C,DEFAULT }

{grid1: !grid }

{Q1:
How many surveys do you take in a year?
!NUM,,,0-100 }

{Q2:
2. Do you think that demonstration surveys are entertaining?
!FLD,
1  Yes
2  No
9  Don't know
}

{!SYS,C,RATING }

{
<table class='qtable'><tr><td colspan='2' class='qtxt'>
On a scale of 1 to 5, how entertaining are the following survey types?
<br/><br/></td></tr>
<table align='center' class='rank'>
<tr><td>  Survey Types </td>
>rep $c=1,...,5;               '' Header for rating scale
<td width="10%">$c </td>
>endrep
</tr>
!DISP }

>rep $a=a,b,c;&b="Medical","Branding","Political";
{Q3$A:
$B
!FLD,
1
2
3
4
5
}
>endrep

{
</table>
!DISP }

{!endgrid }
```

Questionnaire control options

Allowing multi-language characters

Use the [LANGUAGE=…] header statement or {!LANGUAGE …} statement to specify which languages you will be using and use \L<language code> to specify the text for a particular language on a question. The syntax of the !LANGUAGE statement is:

```
{!LANGUAGE CHARACTER_SET=<type> SET=(lang1 lang2 langn) <other options>}
```

Character sets may be EXTENDED_ASCII (European languages), MULTIBYTE (Chinese or Korean), or SHIFT_JIS (for Japanese). The “SET” describes the 2 character language codes such as “EN” for English, “SP” for Spanish, “FR” for French. The first language in the set is the “default” language.

For more information on multiple-language studies, see section 2.5.5 in the Survent manual or Appendix B in this manual.

Response list formats

By default, response lists for single-response CAT and FLD questions with fewer than 200 items are presented in a radio button format. Response lists for multi-response CAT and FLD questions with fewer than 200 items are presented in a check box format. Any response list with 200 or more items usually is presented in a pull-down response box format.

These defaults may be changed by embedding the following compiler commands into the Survent specification.

{!HTML_RADIOBUTTONS=#}

Number of response items below where radio buttons are used for single-response question displays instead of the default pull-down response box with a scroll bar.

{!HTML_CHECKBOXES=#/ALL}

Number of response items below where multi-response questions are displayed with check boxes instead of a pull-down response box with a slide bar (# can be 0 to a very high number.) If this value is set to ALL, then single-response questions use check boxes similar to multi-response questions.

If you choose to go with the default setting of 200, you don’t need a command to set them.

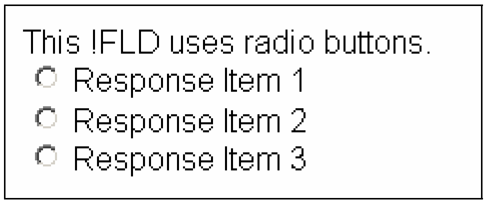
Single-response questions

Radio buttons

The default presentation for a normal (i.e., fewer than 200-item) single-response !FIELD question response list is radio buttons. The specification looks like this:

```
{!HTMLRADIOBUTTONS=200 }  '' Default maximum # of responses to use radio
                             buttons.
{!HTMLCHECKBOXES=200 }    '' Default maximum # of responses to use check
                             boxes.
{Q2A:
This !FLD uses radio buttons.  <br/>
!FLD,B
01 Response Item 1  <br/>
02 Response Item 2  <br/>
03 Response Item 3  <br/><br/> }
```

Will look something like this:


This !FLD uses radio buttons.
○ Response Item 1
○ Response Item 2
○ Response Item 3

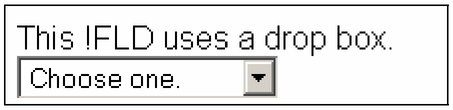
Drop-down boxes

If your response list has more than 200 items (not very likely) or if you have set {!html_radiobuttons= #} to a low number, the response list will be shown as a drop-down box:

The specification looks like this:

```
{!HTMLRADIOBUTTONS=1 }  '' Sets max # of responses to 1 - automatic drop
                           down box.
{!HTMLCHECKBOXES=1 }

{Q2C:
This !FLD uses a drop box.  <br/>
!FLD,B
== Choose one.   ''!FLD comment so there is no default answer.
01 Response Item 1
02 Response Item 2
03 Response Item 3
}
```

Will look something like this:


This !FLD uses a drop box.

Choose one.

NOTE: Since the drop-down box is more labor intensive and not as intuitive to answer as a radio button list, we do not recommend using it unless screen space is an issue.

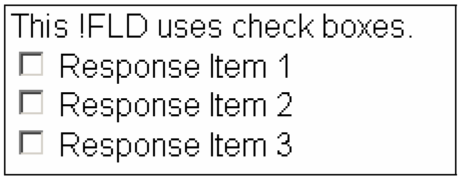
Multiple-response questions:

Check boxes

The default presentation for a normal (i.e., fewer than 200 items), multiple-response !FIELD question response list is check boxes. The specification looks like this:

```
{!HTMLCHECKBOXES=ALL } '' Overrides radio buttons and uses only check
                          boxes.
{Q2B:
This !FLD uses check boxes.  <br/>
!FLD,B
01 Response Item 1  <br/>
02 Response Item 2  <br/>
03 Response Item 3  <br/><br/> }
```

And it will be shown as:


This !FLD uses check boxes.

☐ Response Item 1
☐ Response Item 2
☐ Response Item 3

NOTE*:* If one of the responses is exclusive, meaning that it cannot be answered with some other response (e.g., (-) Don't Know), its response will be shown as a check box. But if it is selected with another response, then upon submitting the screen, an error message will be displayed.

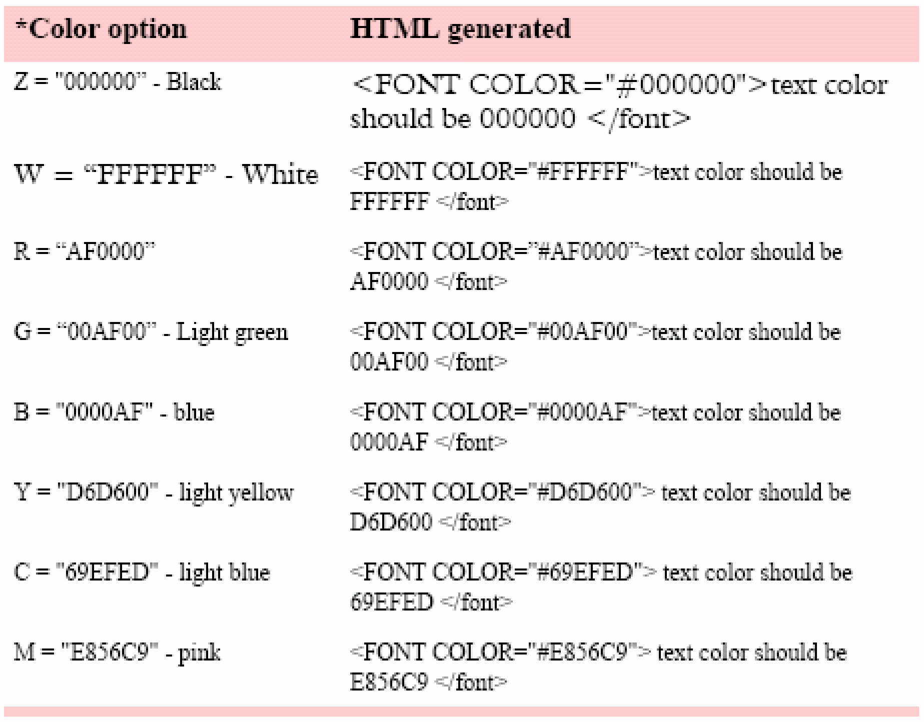
Programming techniques

Using HTML formatting to make text enhancements

Embed HTML code in the questionnaire for formatting or to make use of Web multimedia features such as graphics, sounds or animation. The HTML code is passed through Survent as text to be translated by the browser. For instance, use <B>Bold</B> to bold text, and <BR> to specify that a new line should be started.

Standard CfMC formatting commands may also be used to generate HTML as follows:

Command	HTML	Action
\n	<br>	New line
\b	<b>	Bold
\i	<I>	Italic
\e	</b or /i>	End bold or italic
\w#	<h#>	Header text, # is size and may be from 1 (big) to 5 (small)
\w0	</h#>	Turns off enhancement
\c	<\c(option)>	Sets font color* (color options and the HTML generated are below)



*Color option	HTML generated
Z = "000000" - Black	<FONT COLOR="#000000">text color should be 000000 </font>
W = “FFFFFF” - White	<FONT COLOR="#FFFFFF">text color should be FFFFFF </font>
R = “AF0000”	<FONT COLOR="#AF0000">text color should be AF0000 </font>
G = “00AF00” - Light green	<FONT COLOR="#00AF00">text color should be 00AF00 </font>
B = "0000AF" - blue	<FONT COLOR="#0000AF">text color should be 0000AF </font>
Y = "D6D600" - light yellow	<FONT COLOR="#D6D600"> text color should be D6D600 </font>
C = "69EFED" - light blue	<FONT COLOR="#69EFED"> text color should be 69EFED </font>
M = "E856C9" - pink	<FONT COLOR="#E856C9"> text color should be E856C9 </font>

Creating a color list for rotated grid displays

The compiler directive {!COLOR_LIST abcdefgh} allows you to create a "color list" to assign colors to templates “on the fly” without having to know which question or screen is coming up next. This is used in rotations or questions with skips where you want to maintain a consistent color scheme.

As with many of the HTML commands, this does not get reset when backing up so it should be put inside the grid statement. The keyword to use is:

{!COLOR_LIST <color1> <color2> <color n> }

The colors should be specified as they are in HTML code, as either hexidecimal color codes or names. If you are defining as them as hexidecimal codes, the # sign is required.

So, you would specify codes as:

{!colorlist #ff0000 #0033ff #ffffff #d6d600 #69efed blue red}

The maximum number of colors allowed is 32.

To invoke the colors specified, use:

\cxc	to display the current color
\cxn	to set the "current" color to the next in the list, no display
\cxd	set the "current" color to the next in the list and display it
\cx##	set the "current" to ## (nth color on the list) and display it

The default color is the first color listed.

In the example above, if you used \cx1 the color that would come up is #ff0000.

If it was \cx2 the color would be #0033ff.

The following example shows how to use color_list in rotated attribute questions. This example can be found in CfMC’s demo at: http://demo.cfmc.com/exam81

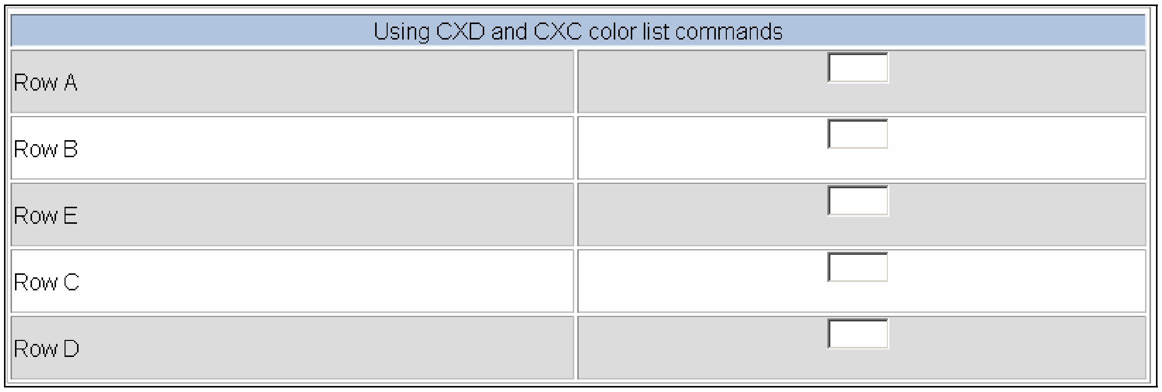
```
{Q30GRD: !GRID }  '' Web page begins here.

'' ===== Q30A - Using color_list features

{!color_list #ffffff #dcdcdc } ''  These two hexadecimal colors will be
used.

{
```

```
<table align="center" border="1" width="60%">
<tr style="background-color:lightsteelblue;"><td colspan='2'
align="center">
Using CXD and CXC color list commands
</td></tr>
!DISP }

{!rotate,s }

>rep $A=A,B,C,D,E;

{Q30A_$A:
<tr>
<td style="background-color:\CXD;"> '' CXD calls the next color from list
and uses it.  CXN calls, but does NOT show color.
Row $A  </td>
<td style="background-color:\CXC;text-align:center;">'' CXC uses the color
called by CXN til a new CXN is called.
\_  </td></tr>
!NUM,B,,0-100 }

>endrep
{!endrotate }

{
</table><br/><br/>
!DISP }

'' ===== Q30B - Using specific colors with color_list

{!color_list yellow grey cyan green red } '' This color_list has 5 colors.
Each represents a color number, 1 through 5.

{
<table align="center" border="1" width="60%">
<tr style="background-color:lightsteelblue;"><td colspan='2'
align="center">
Using CX# color list command to force colors
</td></tr>
```

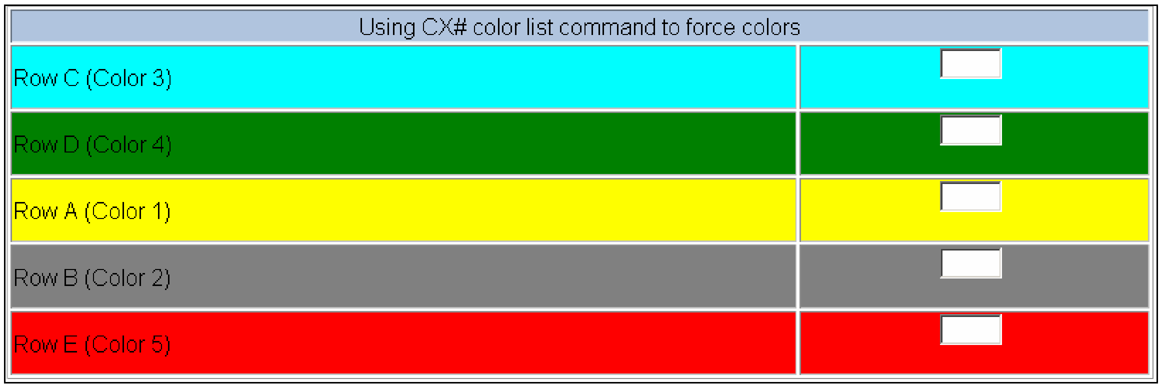
```
!DISP }


{!rotate,s }


>rep $A=A,B,C,D,E;&
      $B=1,2,3,4,5;


{Q30B_$A:
<tr>
<td style="background-color:\CX$B;">  '' CX# forces a specific color for
each row.
Row $A  (Color $B) </td>
<td style="background-color:\CXC;text-align:center;"> '' CXC then uses the
color assigned by CX#.
\_  </td></tr>
!NUM,B,,0-100 }


>endrep
{!endrotate }


{
</table><br/><br/>
!DISP }


'' ===== Q30C - CSS classes and color_list


{
<style type="text/css">  '' An in-spec CSS call.  These two classes will
be used in the next color list.
.class1  \^7B background-color: #00008B; font-family: tahoma; font-size:
12px; font-weight:bold; color: #ffffff;   \^7D
.class2  \^7B background-color: #ffffff; font-family: arial; font-size:
12px; font-style:italic; color: #00008B;   \^7D
</style> '' Use ASCII equivalents for non-CfMC braces.
!DISP }


{!color_list class1 class2 } '' Two classes, defined in the DISP above,
will be alternated.
```

```
{
<table align="center" border="1" width="60%">
<tr style="background-color:lightsteelblue;"><td colspan='2'
align="center">
Using CSS Classes with Color List
</td></tr>
!DISP }

{!rotate,s }

>rep $A=A,B,C,D,E;

{Q30C_$A:
<tr class="\CXD"> '' Assigns the class from the stylesheet elements above.
<td> Row $A </td>
<td style="text-align:center;"> \_  </td></tr>
!NUM,B,,0-100 }

>endrep
{!endrotate }

{
</table><br/><br/>
!DISP }

{!ENDGRID }
```

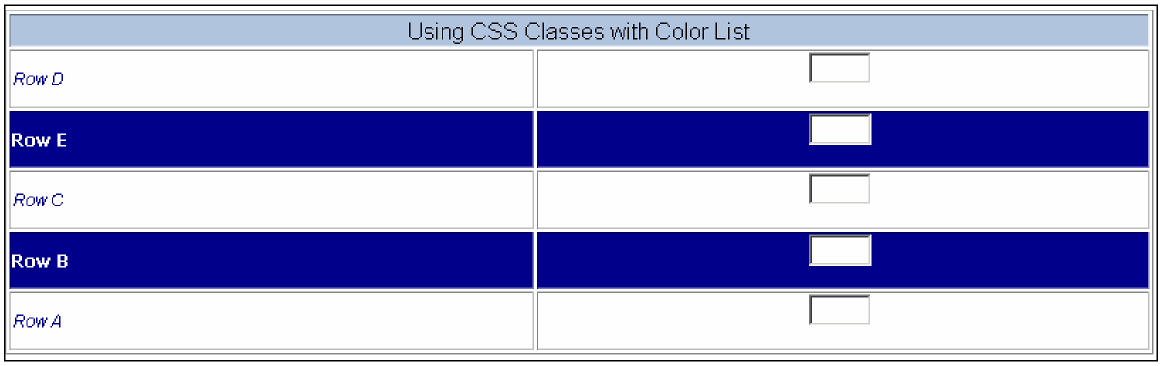
And it will be shown as:



Using CXD and CXC color list commands	
Row A	☐
Row B	☐
Row E	☐
Row C	☐
Row D	☐



Using CX# color list command to force colors	
Row C (Color 3)	☐
Row D (Color 4)	☐
Row A (Color 1)	☐
Row B (Color 2)	☐
Row E (Color 5)	☐



Using CSS Classes with Color List	
Row D	☐
Row E	☐
Row C	☐
Row B	☐
Row A	☐

You can specify other HTML text controls by using HTML language. Consult an HTML manual to learn all of the options available. Or, if time is limited, use an HTML editor to get a certain look and feel. Then copy the HTML source from that document to your questionnaire specifications.

Browsers read text in continuous mode and wrap according to the browser window size; they do not evaluate normal line breaks in script text. Use \n or the <br> HTML code to insert a line break, or use the special HTML compiler options described above to add a <br> after each text line. To line up a response list vertically, add \n or <br> to the end of each response item in the question.

Grids, Endgrids, & Endgrid Errors

Grids allow you to program multiple questions on a webSurvent page. In verson 8.1, we've added the ability to generate Endgrid Errors, so you can enforce complex backend checks without making the respondent leave and return to the page.

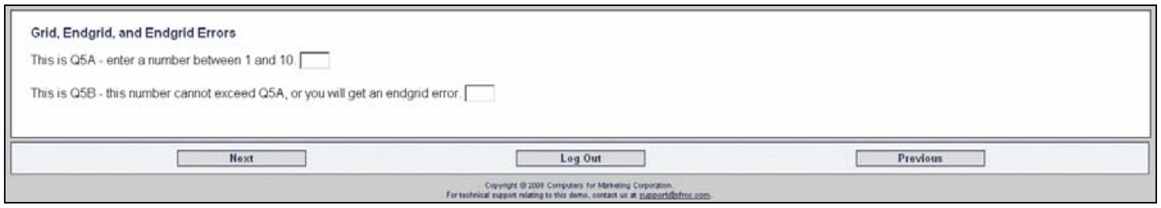
Grids and endgrids are used in your spec to determine the beginning and end of each rendered web page. By default, webSurvent assumes every question will be displayed on a separate page, unless you encase them in a grid.

In previous versions of webSurvent, you were required to build separate HTML pages to handle your backend checks, and then return them to the previous grid. In 8.1, you can also program !endgrid_errors on your page, to enforce questionnaire logic and backend checks on the same screen - the error will appear in the same location as your griderror when the respondent tries to submit. You can also assign levels to these endgrid errors, to enforce logic sequentially, rather than trigger every error simultaneously once the page is submitted.

Below is an example of its use within a spec:

```
{!html_griderror_prefix=<div style="text-align:center;width:100%;color:#ff0000;">}

'' To format griderror and endgrid error.  Must be set before grid.

{!html_griderror_suffix=</div> }


{Q5AGRID: !GRID }  '' Web page begins here.


{Q5A:

This is Q5A - enter a number between 1 and 10:  \_  <br/><br/>

!NUM,,,0-10 }


{Q5B:

This is Q5B - if it exceeds Q5A, you'll receive an endgrid error at the top of this
screen:  \_   <br/><br/>

!NUM,,,0-10 }


{egerror:

!IF x(Q5B)>x(Q5A)'' Endgrid error logic works for questions in the same grid.

The endgrid_error appears at the start of the page to tell you that Q5B cannot
exceed Q5A.

!endgrid_error } ''  Endgrid_Error command - must be used within grid.


{!ENDGRID }'' (Web page ends here.)
```

And it will be shown as:



Grid, Endgrid, and Endgrid Errors

This is Q5A - enter a number between 1 and 10.

This is Q5B - this number cannot exceed Q5A, or you will get an endgrid error.

Copyright © 2008 Computers for Marketing Corporation.
For technical support relating to this demo, contact us at support@cfmc.com.

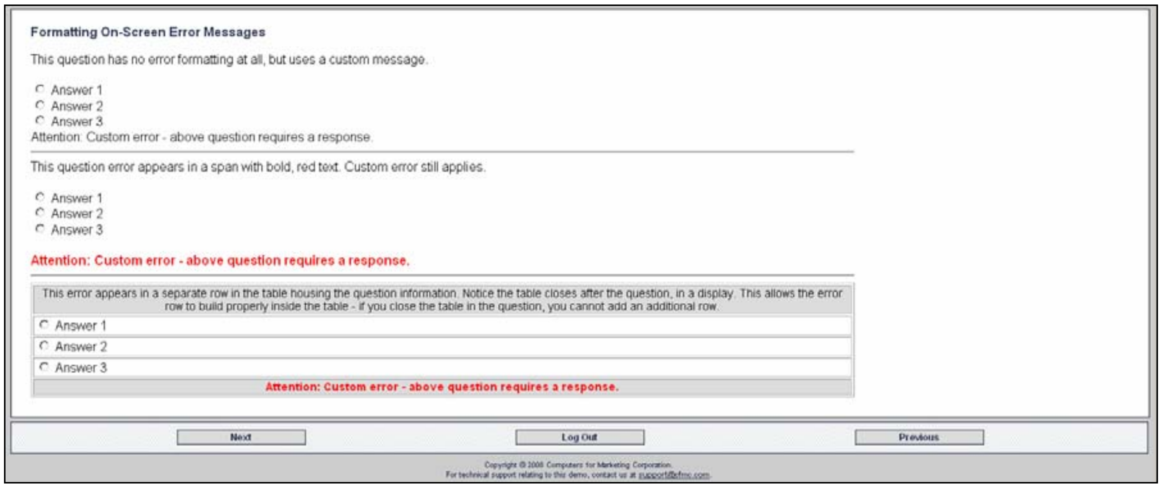
Presenting error messages

In webSurvent, error messages must be formatted so that they stand out visually. Error placement and customization are crucial for providing the respondent with feedback when they answer a question improperly. The example below show you how to format on-screen error messages in webSurvent.

```
'' == Q6 Grid


{Q6GRD: !GRID }  '' Web page begins here.


'' ===== Q6A


{!html_error_prefix= }
{!html_error_suffix= }
{!error_msg 1758 Custom error - above question requires a response. Can be
done in grid. }


{Q6A:
This question has no error formatting at all, but uses a custom message.
<br/><br/>
!FLD,
1 Answer 1 <br/>
2 Answer 2 <br/>
3 Answer 3 <br/>
}


'' ===== Q6B


{!html_error_prefix=<br/><span style="color:#ff0000;font-weight:bold;"> }
{!html_error_suffix=</span>}


{Q6B:
```

```
<br/><hr width="75%" align="left">

This question's error appears below, in a span with bold, red text. Custom
error still applies.   <br/><br/>

!FLD,

1 Answer 1 <br/>

2 Answer 2 <br/>

3 Answer 3 <br/>

}


'' ===== Q6C

'' Prefix below opens a new row to hold error.

{!html_error_prefix=<tr><td style="background-color:#DCDCDC;font-
size:14px;font-weight:bold;color:#ff0000;text-align:center;"> }

{!html_error_suffix= </td></tr> } '' Closes row and cell so table can
continue.


{Q6C:

<br/><hr width="75%" align="left">

<table border="1" width="75%" align="center"><tr><td style="text-
align:center;background-color:#DCDCDC;font-size:14px;">

This error appears in a separate row in the table housing the question
information.  Notice the table closes after the question, in a display.
This allows the error

row to build properly inside the table - if you close the table in the
question, you cannot add an additional row.

</td></tr><tr><td>

!FLD,

1 Answer 1 </td></tr><tr><td>

2 Answer 2 </td></tr><tr><td>

3 Answer 3 </td></tr>

}


{

</table><br/><br/> '' Must close table after question so error can appear
within table structure.

!DISP }

{!ENDGRID }

{!-error_msg 1758 } '' Resets error to default in msgfile. Can only be
done after grid.
```

If a respondent fails to answer a question or answers a question inappropriately, an on-screen error will appear when they attempt to submit the page. These errors have no predetermined formatting, so you must use the !html_error commands to apply HTML and/or styles to the text, so that it stands out and quickly shows the respondent where the issue is. Below is what the respondent will see when these errors are in action when submiting the page without answering the three questions as seen here.


Formatting On-Screen Error Messages

This question has no error formatting at all, but uses a custom message.

- Answer 1
- Answer 2
- Answer 3

Attention: Custom error - above question requires a response.

This question error appears in a span with bold, red text. Custom error still applies.

- Answer 1
- Answer 2
- Answer 3

Attention: Custom error - above question requires a response.

This error appears in a separate row in the table housing the question information. Notice the table closes after the question, in a display. This allows the error row to build properly inside the table - if you close the table in the question, you cannot add an additional row.
Answer 1
Answer 2
Answer 3
Attention: Custom error - above question requires a response.

Copyright © 2008 Computers for Marketing Corporation.
For technical support relating to this demo, contact us at support@cfmc.com.

Customizable errors on !endgrids

Use !endgrid_error to present specific error messages based on a condition when the grid is submitted. You can use as many of these as you choose inside a grid.

```
{
!if condition
error text
!endgrid_error }
```

There are two methods to control the presentation of these messages. The first is whether to show all errors at once or to only show one at a time. The default is to show all at once {!show_all_endgrid_errors} but it can be turned off with {!-show_all_endgrid_errors} which would show only one error at a time.

You can set a level on the !endgrid_error command. This will allow you to present errors in a specific order or to show specific groups of errors together. The error with the lowest level will be shown first, the error with next highest level would be shown next. Any errors without a level assigned would be shown last.

```
{g2:
Start of grid
!grid }


{Q2a:
q2a Title text
!fld
1 one
2 two
3 three }

{!if not(Q2a(1))
This is endgrid error 2a. Q2a is not a 1.
!endgrid_error level=2 }

{Q2b:
q2b  Title text
!fld
1 one
2 two
3 three }

{!if not(Q2b(1))
This is endgrid error 2b. Q2b is not a 1.
!endgrid_error level=2 }

{Q2c:
q2c Title text
!fld
1 one
2 two
3 three }

{!if not(Q2c(1))
This is endgrid error 2c. Q2c is not a 1.
!endgrid_error level=1 }

{!endgrid}
```

In the example above, if there were three errors on the page, the error for q2c would be shown first. Once that was corrected the errors for q2b and q2a would be shown.

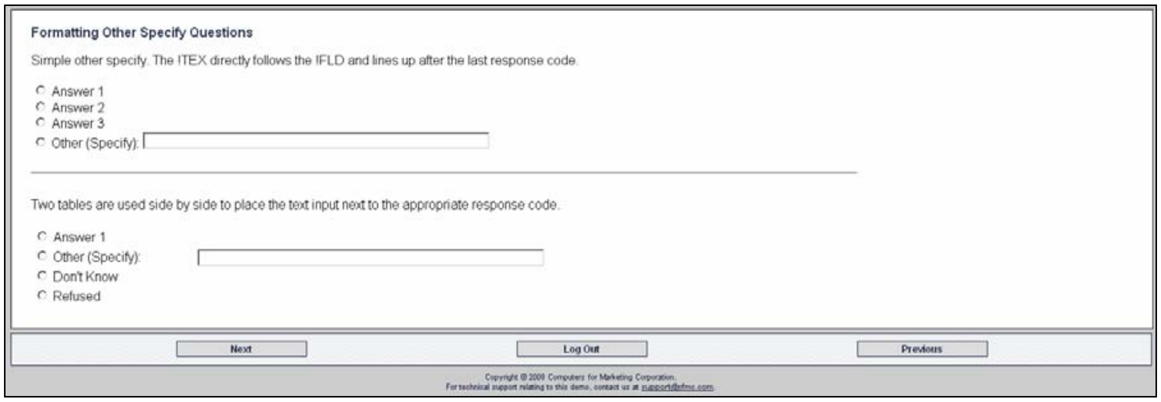
Formatting Other Specify

Other Specify actually requires you to build two questions, a !FLD and a !VAR/!TEX, within a grid. Formatting an Other Specify according to client specifications can be tricky, since in some cases, it requires you to place the text input between response options. This example shows you how to do this with tables.

```
'' == Q7 Grid


{Q7GRD: !GRID }  '' Web page begins here.


'' ===== Q7A


{Q7A:

Simple other specify.  The TEX directly follows the FLD and lines up after
the last response code.   <br/><br/>

!FLD,B

1 Answer 1 <br/>

2 Answer 2 <br/>

3 Answer 3 <br/>

4 Other (Specify): }


{Q7AOTH:

\_ <br/><br/>

!TEXT,B,1,50,1000 }




{Q7B:

<hr width="75%" align="left"><br/>

Two tables are used side by side to place the text input next to the
appropriate response code. <br/><br/>

<table width="15%" align="left" cellpadding="0" border="0"> '' First
table, holds response list.

<tr><td align="left" valign="top" height="22" valign="middle">

!FLD,B

1 Answer 1 </td></tr><tr><td align="left" valign="top" height="22"
valign="middle">

2 Other (Specify):   </td></tr><tr><td align="left" valign="top"
height="22" valign="middle">

8 Don't Know  </td></tr><tr><td align="left" valign="top" height="22"
valign="middle">

9 Refused </td></tr> }
```

```
{Q7BOTH:
</table>
<table width="85%" align="left" cellpadding="0" border="0"> '' Second
table, holds text input.
<tr><td height="22"></td></tr>
<tr><td align="left" height="22" valign="top">
\_
</td></tr>
!TEXT,B,1,50,1000 }

{   '' Remaining rows not in use.
<tr><td height="22"></td></tr>
<tr><td height="22"></td></tr>
</table>
!DISP }

{!ENDGRID }
```

And it will be shown as:


Formatting Other Specify Questions

Simple other specify. The ITEX directly follows the IFLD and lines up after the last response code.

- Answer 1
- Answer 2
- Answer 3
- Other (Specify):

Two tables are used side by side to place the text input next to the appropriate response code.

- Answer 1
- Other (Specify):
- Don't Know
- Refused

Next
Log Out
Previous
Copyright © 2009 Computers for Marketing Corporation.
For technical support relating to this demo, contact us at support@cfmc.com.

Storing data in a hidden input using VAR,H

!VAR,H – This is used to create a hidden input tag onscreen. This question subtype can be used to gather data filled in by some other program such as JavaScript, Shockwave, Flash or another browser-based programming language. This can be anything that will allow a respondent to give an answer.

With JavaScript and !var,h, you can pass data to a hidden input tag. You can make a clickable image that actually writes data or an image map that writes to data.

So,

```
{question:
!var,h,2,0}
```

Would create a hidden input tag on the screen:

```
<input type="hidden" name="var_question" value"">
```

Keep in mind that because these questions are hidden onscreen, there is no way for webSurvent to present an understandable error message to the respondent. Since that is the case, all error checking must be done by the program that is filling in the question.

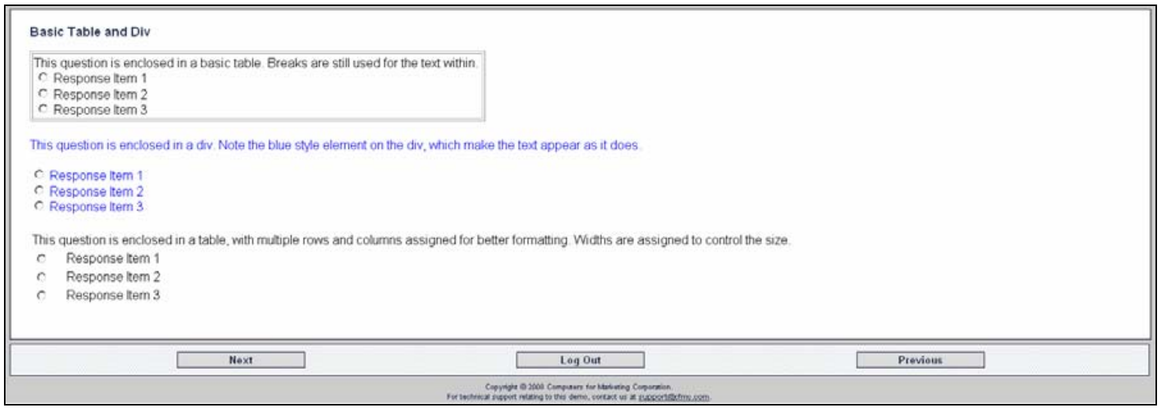
Tables and Divs

Tables and Divs are important HTML tags used to format your web surveys with clean, concise designs. They can be used to format almost any question, from invisible grids for simple formatting to complex multi-celled structures like attribute lists or rating scales. Style elements like width percentages allow you to auto-wrap text in a way that will allow for a consistent look on any screen resolution or window size.

The three questions below are encased in HTML tags, so you can better control the layout of the text via table or style elements. One benefit is that tables and divs can be assigned a horizontal percentage of the user's browser window. Assigning width="100%" or style="width:100%" will expand to cover the entire browser window, regardless of resolution. For web surveys, where static appearance and ease of use across all sample is paramount, these tags are invaluable.

```
{!GRID }


{Q3_1A:
<table border="1"><tr><td>   '' This begins the table, row (tr), and cell
(td).  We've set a table element to show border.
This question is enclosed in a basic table.  Breaks are still used for the
text within.   <br/>
!FLD,B
01 Response Item 1 <br/>
02 Response Item 2 <br/>
03 Response Item 3 </td></tr></table><br/>  '' After the question, we
close the cell, row, and table in opposite order.
}


{Q3_1B:
<div style="color:blue;">  '' Div is opened.  Unlike table, it has no
built-in style elements - you must assign them.
This question is enclosed in a div.  Note the blue style element on the
div, which make the text appear as it does.   <br/><br/>

!FLD,B

01 Response Item 1 <br/>

02 Response Item 2 <br/>

03 Response Item 3 </div><br/>  '' Div is closed.

}


{Q3_1C:

<table width="75%"><tr><td colspan='2'>  '' Width is controlled for table.
First cell will span two columns.

This question is enclosed in a table, with multiple rows and columns
assigned for better formatting.

Widths are assigned to control the size.

</td></tr><tr><td width="4%">  '' Close previous cell and row, open new
row and cell for radio button only.

!FLD,B

01 </td><td width="96%"> Response Item 1 </td></tr><td><tr>  '' widths
cascade, so only need to be assigned once.

02 </td><td> Response Item 2 </td></tr><td><tr>

03 </td><td> Response Item 3 </td></tr></table><br/>  '' after last
response, close table.

}


{!ENDGRID }
```

And it will be shown as:



Basic Table and Div

This question is enclosed in a basic table. Breaks are still used for the text within.
○ Response Item 1
○ Response Item 2
○ Response Item 3

This question is enclosed in a div. Note the blue style element on the div, which make the text appear as it does.

○ Response Item 1
○ Response Item 2
○ Response Item 3

This question is enclosed in a table, with multiple rows and columns assigned for better formatting. Widths are assigned to control the size.
○ Response Item 1
○ Response Item 2
○ Response Item 3

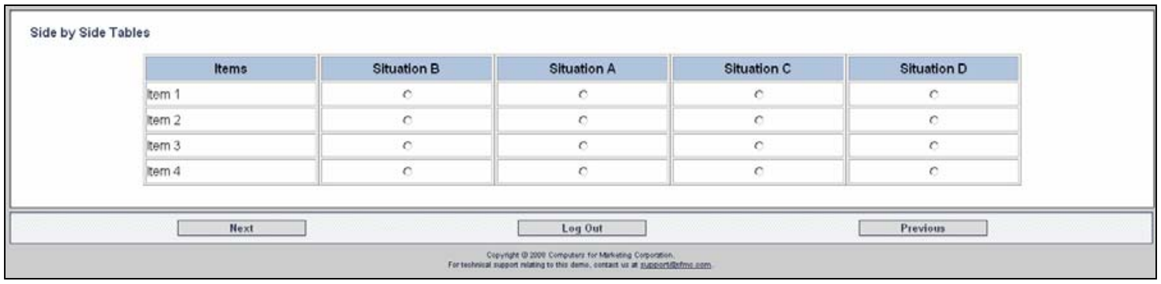
Copyright © 2008 Computers for Marketing Corporation.
For technical support relating to this demo, contact us at support@cfmc.com.

Below is an example of side by side tables in a grid.

```
{!GRID }


{
<table width="80%" align="center"><tr><td align="center">  '' Container
table to hold other tables.
<table border="1" width="20%" align="left">
<tr><th style="background-color:lightsteelblue;" height="30"> Items
</th></tr> '' First column, holds actual attributes/items.
>rep $A=1,2,3,4; '' Repeat for each attribute/item.
<tr><td height="30"> Item $A </td></tr>
>endrep
</table>
!DISP }


{!html_error_prefix=<tr><td height="20" style="background-
color:#dcdcdc;color:#ff0000;font-weight:bold;">} '' Error formatting.
{!html_error_suffix=</td></tr>}


{!ROTATE,S } '' To rotate the tables holding each question.
>rep $B=A,B,C,D;  '' Repeat for 4 variables.


{!GROUP } '' To group the questions and displays below them.


{Q3_3$B:
<table border="1" width="20%" align="left"><tr><th style="background-
color:lightsteelblue;" height="30">
Situation $B
</th></tr><tr><td height="30" align="center">
!FLD,B
1 </td></tr><tr><td height="30" align="center">
2 </td></tr><tr><td height="30" align="center">
3 </td></tr><tr><td height="30" align="center">
4 </td></tr>
}


{
```

```
</table> '' Closes after question, so error row builds within question
table.
!DISP }


>endrep
{!ENDROTATE }


{
</td></tr></table>   '' Closes off container table.
!DISP }


{!ENDGRID }
```

And it will be shown as:


Side by Side Tables

Items	Situation B	Situation A	Situation C	Situation D
Item 1	○	○	○	○
Item 2	○	○	○	○
Item 3	○	○	○	○
Item 4	○	○	○	○

Copyright © 2009 Computers for Marketing Corporation.
For technical support relating to this demo, contact us at support@cfmc.com.

Attribute lists

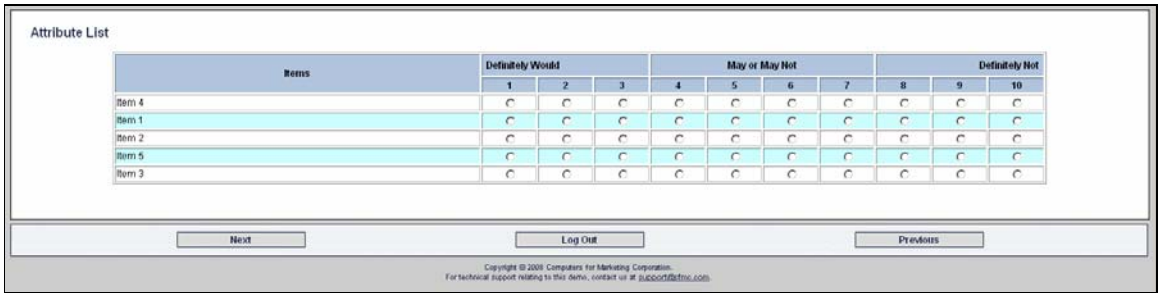
Below is an example of a table which has 11 columns and 7 rows: 2 rows for the header, and 5 rotated rows for the items, which are contained in a repeat. Color List is used to alternate colors between the rotated rows. !Error_msg commands can be used to customize the errors.

```
{!GRID }


{!ERROR_MSG 7222 Attention: }
{!ERROR_MSG 1758 Please select a ranking for the item above. }


{
<table width="85%" border="1" align="center" style="font-size:12px;">
 <tr style="background-color:lightsteelblue;">
  <th rowspan="2" width="40%" style="font-weight:bold;">'' Rowspan
accounts for both rows of header.
```

```
 Items
  </th>
  <th colspan="3" width="18%" style="text-align:left;padding:5;font-
weight:bold;"> Definitely Would </th>
  <th colspan="4" width="24%" style="text-align:center;padding:5;font-
weight:bold;"> May or May Not </th>
  <th colspan="3" width="18%" style="text-align:right;padding:5;font-
weight:bold;"> Definitely Not </th>
 </tr>
 <tr style="background-color:lightsteelblue;">
>rep $A=1,...,10;
  <th width="6%"> $A  </th> '' Header numbers in repeat.
>endrep
 </tr>
!DISP }

{!HTML_ERROR_PREFIX=<tr><td bgcolor="#DCDCDC" colspan="11"
style="color:#FF0000;text-align:center;font-weight:bold;padding:2;">}
'' Colspan of 11, so error row spans entire table.  Styled to stand out.
{!HTML_ERROR_SUFFIX=</td></tr>}

{!COLOR_LIST #ccffff #ffffff }  '' Color List used for rows (see Color
List example for more details).

{!ROTATE,S }

>rep $A=1,...,5;

{Q4$A:
<tr bgcolor="\CXD">
 <td style="text-align:left;">
Item $A
 </td><td align="center">
!FLD,
01 </td><td align="center">
02 </td><td align="center">
03 </td><td align="center">
04 </td><td align="center">
05 </td><td align="center">
```

```
06 </td><td align="center">
07 </td><td align="center">
08 </td><td align="center">
09 </td><td align="center">
10 </td></tr>
}

>endrep
{!ENDROTATE }

{
</table><br/>
!DISP }

{!ENDGRID }

{!-ERROR_MSG 1758 } '' Resets error message to default.
```

And it will be shown as:



Attribute List

Items	Definitely Would			May or May Not				Definitely Not		
	1	2	3	4	5	6	7	8	9	10
Item 4	○	○	○	○	○	○	○	○	○	○
Item 1	○	○	○	○	○	○	○	○	○	○
Item 2	○	○	○	○	○	○	○	○	○	○
Item 5	○	○	○	○	○	○	○	○	○	○
Item 3	○	○	○	○	○	○	○	○	○	○

Copyright © 2008 Computers for Marketing Corporation.
For technical support relating to this demo, contact us at support@cfmc.com

Using !ROTATE

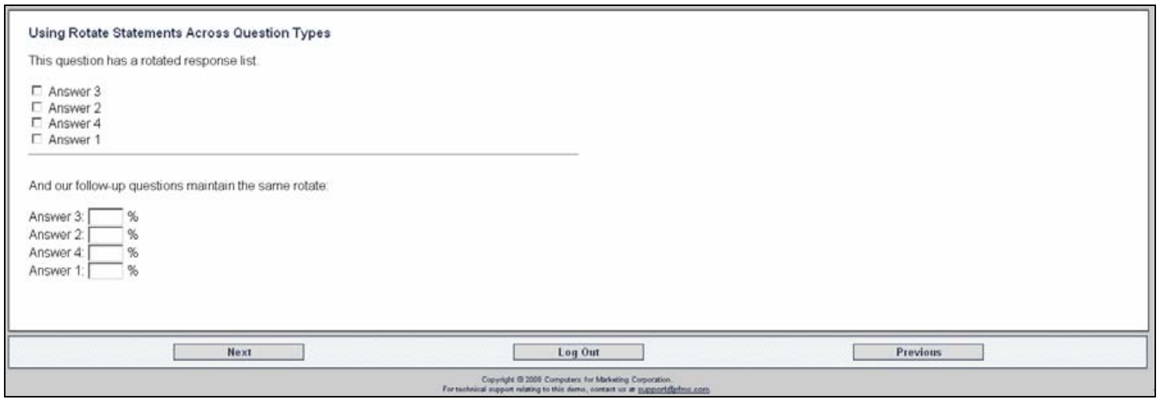
This command allows you to rotate response items, questions, or entire sections of your questionnaire. You can also store your rotate seed to control later rotations, and group or fix responses or questions to your clients' specifications.

Rotate commands can be used within !FLD questions to rotate response codes. With this method, you can save a rotate seed and reuse it later with other question types, assuming you use the same number of items for each rotate.

```
'' == Q31 Grid


{Q31GRD: !GRID }
```

```
{ROTSEED: HIDE
!VAR,N,10,10 }   '' Hold the rotate seed for next questions.

{Q31A:
This question has a rotated response list. <br/><br/>
!FLD,,4
!ROTATE,S,1,ROTSEED  '' Rotate is used instead of FLD,R.  ROTSEED is
saved.
1 Answer 1 <br/>
2 Answer 2 <br/>
3 Answer 3 <br/>
4 Answer 4 <br/>
!ENDROTATE
}

{
<hr width="50%" align="left">  '' Line to separate questions.
<br/>
And our follow-up questions maintain the same rotate: <br/><br/>
!DISP }

{!ROTATE,S,1,[ROTSEED] }   '' Uses the same ROTSEED as Q31A.
>rep $A=1,2,3,4;

{Q31B$A:
Answer $A: \_   %  <br/>
!NUM,,,0-100 }

>endrepeat
{!ENDROTATE }

{
<br/><br/>
!DISP }

{!ENDGRID }
```

And it will be shown as:



Using Rotate Statements Across Question Types

This question has a rotated response list.

☐ Answer 3
☐ Answer 2
☐ Answer 4
☐ Answer 1

And our follow-up questions maintain the same rotate:

Answer 3: ▭ %
Answer 2: ▭ %
Answer 4: ▭ %
Answer 1: ▭ %

Copyright © 2008 Computers for Marketing Corporation.
For technical support relating to this demo, contact us at support@cfmc.com.

CHAPTER 4

SETTING UP A STUDY

For webSurvent, first determine whether the study is closed or open. A **closed study** requires respondents to enter a pre-designated name and password to access the interview. An **open study** has no password protection, so anyone can get in and take it. The program generates passwords for each open study respondent to allow them to suspend and resume the interview. Keep in mind, when you have an open study, respondents can get into the survey multiple times.

Making a subdirectory for study files

Create a directory for the study. You may create this directory anywhere. Generally you will want it named with the study name and in a standard area. CfMC provides the study area ${CFMC}websurv/studies if you wish to use that. This is where you will create your study files, compile the scripts, and create the sample files that you need.

Creating a raw phone sample file for names and passwords

WebCATI uses CfMC phone sample files to store phone numbers, names and any other information associated with a pool of potential respondents for a study. Use fonebuld as specified in *Chapter 6 of the Survent manual* to make a sample file for webCATI.

CfMC has also adapted the phone sample file for webSurvent as well. In this case, we use a unique username and password combination as an identifier instead of a phone number.

For webSurvent in "closed study" mode, specific user names are used to track and support a study. If doing an open study, a password is generated for each respondent. Whether the study is open or closed, a phone sample file for the study (<study>.fon) plus two phone index files (<study>.fnx – the default for the phone number and <study>.fny – the index for the password) must exist. The phone file for an open study is initially empty while the phone file for a closed study has a record containing a unique password and a username (either unique or generic) for each potential respondent.

This is a layout example of a raw ASCII phone sample file for webSurvent (this is a compressed view of the phone sample file for documentation purposes, so it will look different on your screen.)

```
0         1         2         3         4         5         6         7

1234567890123456789012345678901234567890123456789012345678901234567890123456678684
159999999                                         abcdefg             Joe User
```

01.10 – Phone number (use unique dummy 800 numbers if real phone numbers

don't exist, e.g., "8000000001" and so on)

51.20 – Alphanumeric password 5-20 characters, left-justified in the field

71.30 – Respondent Name, left-justified in the field. The name can be 5-30 characters.

Creating the index for the password in webSurvent (.fny)

The phone file for webSurvent is created basically the same way as any other phone file. The difference is that a secondary index is required. You must define an index file that points to the password location for both open and closed studies. The password must appear in the raw phone file starting in column 51, with a maximum width of ten columns. The FONEBULD index statement looks like this:

```
index 51,20
```

Reformatting raw sample file with WSMOVER

Since certain information in the phone record is required to be in certain fields, "wsmover" has been devised as one way to reformat your raw sample file. You may also use the file "mover.spx" in the ${CFMC}survent directory if you have special custom conversion requirements and want more control, or use a customized 'mentor' script.

The raw sample input must be ASCII text of any desired record length and each field within the record must be of a fixed location and width. Tabs and other special characters are not allowed (Note: the COPYFILE utility or mentor will convert tab or comma delimited files to fixed format for you).

WSMOVER will also create a file of solely dummy sample for testing purposes.

Before running WSMOVER, please be aware of the location and width of each data field.

To run WSMOVER, type:

```
wsmover ←
```

Enter an input sample file name at the prompt, or return if you have no input:

```
Enter <input filename> or ← (enter) for none
```

At the next prompt, enter an output ASCII file name:

```
Enter <output filename> or ← (enter) to exit
```

If your output file already exists, you will have the option to add to the existing file or overwrite it.

The next prompt deals with auto-generated test sample. Up to 99,999 test fone records can be generated with the following attributes:

Phone #: 888-11<serialized 5-digit-#>

Username: Respondent

Password: test<serialized 5-digit-#>

The program prompts for the number of records to create:

```
How many test fone records do you want? (100) n for none
```

Enter "n" if you do not want any test records. If you define a positive value at this prompt, you will be prompted for a number to start the serialization of the phone records:

Enter starting number for serializing test fone records or press ← <Enter> for 1.

Pressing Enter will start serializing phone numbers. The first record will have the phone number "8881100001", the second will have "8881100002" and so on. If you add sample to an existing file that had test phone numbers added to it previously, be sure to define a starting number that will prevent the duplication of phone numbers in the file.

If you have defined an input file, you will be prompted for information regarding its layout. First, the phone number location (Starting at column 1 and up to a width of 20 in the output):

> Enter starting location of phone number or location.width if width is not equal to 20 or ← <Enter> for no phone number
>
> A phone number does not have to exist in your input record; it may be generated for you.

A single return will give you the following prompt:

> A phone number has to exist in the record.
>
> We can generate fake phone numbers with the structure "8000001001"
>
> Enter starting number for bogus numbers (1001).
>
> This time, the first record will have the phone number "8000000001", the second will have "8000000002" and so on. Once again, if you are adding sample to an existing file that had bogus phone numbers added to it before, be sure to define a starting number that avoids duplicate phone numbers in the file.

The next prompt deals with the password (51.20 in the output):

> Enter width of password or press ← <Enter> for 6 (max=20).
>
> The password may exist in your input record, it may be generated for you, or you may use a combination of the two options. You also may add a fixed identifier of one to two characters to the password. This may be useful if you want to identify sample quickly.

The next two prompts will appear in a repeating series until your password width has been used up:

> Enter starting location of field 1 of the password, enter "i" to define a fixed identifier, or press ← <Enter> to have the password randomly generated for you.
>
> Enter width of field 1 of the password or press ← <Enter> for 6.
>
> This will go on to field 2, and so on, until the entire width of your designated password field has been defined.

The next prompt is for username definition (71.30 in the output):

> Enter Username information: (30 characters max).
>
> Enter one of the following:
>
> > <startcol>.<wid> - if username is in sample
> >
> > r - generate username "Respondent" for all records
> >
> > e - generate username from first part of email address
>
> (For example, "labufadora@cfmc.com" generates username "labufadora") If the e-mail address used in option e has more than 20 characters before the "@", the first 20 characters will be used.

The next prompts deal with the e-mail address (91.60 in the output):

> Enter <startcol.wid> for e-mail address location or <Enter> for none (maxwid=60)

If option e was defined in the previous prompt, this field must be defined. An optional check of the e-mail address structure ("<name>@<something>.<com>") may be performed as well by replying to the next prompt:

> Would you like to check the e-mail addresses? (y/(n)) ←

If you choose "y" a list of bad e-mail addresses will be presented and you will be prompted to continue or exit.

The next set of prompts deals with any other locations that you want to move from one file to another. They may be added to any column between 151 and 950 in your output; just be careful not to overlap. You may add up to 5 "other" fields:

Enter <startcol.wid> for first "other" location in sample file or

← <Enter> if no more locations.

Enter the starting location in for first "other" field in output file.

NOTE: This must be a number above 161 and may not extend beyond column 950.

Press ← <Enter> for 161

For each field, the receiving location will default to the next open field. A summary table will appear next, giving you the opportunity to make any changes to what you have entered:

A Input file name: inname

B Output file name: outname

C Number of Test Records: 100 Start at: 1

D Password Info: Total Width: 6

Part 1 Generated width:6

E Username Info: All will have username "Respondent"

F Phone Number Info: Start Phone Numbers at: 1001 (e.g., 800-1001)

G Email Location: None Defined

H First Other Location: from:13.67 to:91

I Second Other Location: Not Used

J Third Other Location: Not Used

K Fourth Other Location: Not Used

L Fifth Other Location: Not Used

Enter the letter of field(s) to change or press <Enter> to continue -->

Pressing ← (enter) will start the process of creating the file. You will see one (1) dot per 1000 records processed. The output should be ready for input to FONEBULD or the MAKEFONE utility.

Using FONEBULD or MAKEFONE to create a phone sample file

You can use FONEBULD for webCATI or MAKEFONE for webSurvent to create the phone sample file. See the Survent manual chapter 6 for more info on FONEBULD.

With MAKEFONE, specify a study code and the name of the raw ASCII sample file that contains names and passwords to build a file for a closed study. Specify only the study code to build a file with no records in it for an open study.

The MAKEFONE command has the following format:

```
makefone <studycode> <rawfilename> [text length]
```

The Text length is the length of the phone text area which may be between 200 and 4900 columns. The default is 200 columns. Leave this area blank, or use it to pre-load any information about prospective respondents. For example, to make an empty phone sample file for use with an open study called "bank," you would type the following:

```
makefone bank
```

If bank is a closed study and the ASCII file containing names and passwords, it is called bank.raw, type:

```
makefone bank bank.raw
```

To specify 400 characters of phone text area:

```
makefone bank 400 (if bank is an open study)
makefone bank bank.raw 400 (if bank is a closed study)
```

Compiling your questionnaire and creating study files

At this point, you'll want to compile your questionnaire. Compiling will result in the creation of the questionnaire (spec file, QFF) and a quota file (QUO), as well as the backup spec files and summary information (.CHK for a list of data locations, .SUM for a question list, .HRD for a readable hardcopy listing, .DB for variables for the reports, and .QSP for backup questionnaire specifications).

NOTE: Save the .qpx file and all of these files for each version you put in the field. Without them you may not be able to recover mismatched data if you make many changes.

Using the LOADSTUDY utility to move files into the active CFMC area

Use the LOADSTUDY utility to move the .QFF questionnaire file, .QUO quota file, and and the .FON/.FNX/.FNY phone sample file and index files you created for names and passwords to their appropriate ${CFMC} directories. NOTE: You must shut down the study before using LOADSTUDY to load a new version.

If the study is active under the CfMC server, and there are no sessions running, LOADSTUDY will prompt for an LDEV and deactivate the study. At the end of the run, the user will be asked whether or not they want to immediately activate the study.

To activate LOADSTUDY, type:

```
loadstudy <studycode> ←
```

NOTE: Loadstudy will not copy files if the study is currently active. If the study is active, you must deactivate it so that the files are no longer in use by the server. Refer to *Deactivating a Study section in Chapter 9* of this manual.

These are the functions of LOADSTUDY:

It copies <studycode>.qff to the ${CFMCQFL} directory (default ${CFMC}qff/).

It copies <studycode>.quo to the ${CFMCQUOTA} directory (default ${CFMC}quota/), or loads the quotas from the prior quota file into the new file before copying.

It copies <studycode>.db to the ${CFMCDB} directory for use with CfMC data utilities default ${CFMC}data/).

It copies <studycode>.fon, <studycode>.fnx and <studycode>.fny to the ${CFMCFONE} directory (default: ${CFMC}fone/).

It updates the study's suspended files to match the current/new questionnaire if requested. Files cannot be updated to match, a warning is generated.

If a previous version of the study exists, LOADSTUDY makes a backup copy of the files with the name <studycode>.<ext>.<yymmdd> (e.g. cars.qff.040307). It logs all copies to the loadstudy.log file in the ${CFMCQFL} (default:${CFMC}qff/) directory.

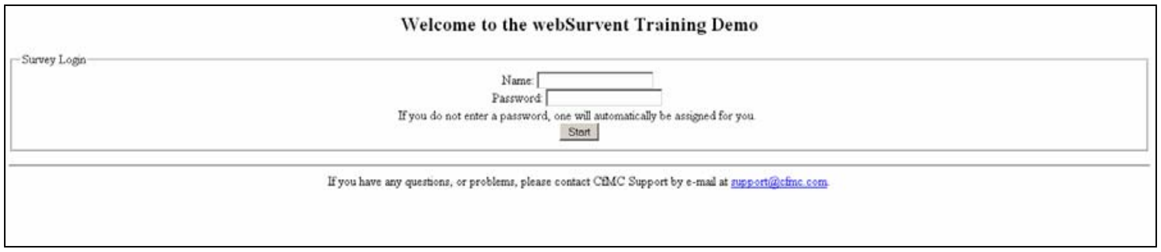
CHAPTER 5

HTML FILES USED

The webSurvent index.html file

This is the first screen that respondents see when they access a CfMC Web questionnaire. WEBUTIL uses the files that exist in ${CFMC}websurv/archive/webutil to create the HTML files for the study. After WEBUTIL creates them, you may reformat them using HTML code.

Edit the original index.html file in the WEBUTIL directory for a default look-and-feel for all your studies. WEBUTIL makes the necessary wording changes on a study-by-study basis. Here is an example of an index page that you can put up on the Internet:



Welcome to the webSurvent Training Demo

Survey Login

Name:
Password:
If you do not enter a password, one will automatically be assigned for you.
Start

If you have any questions, or problems, please contact CfMC Support by e-mail at support@cfmc.com

The HTML source code for the above screen appears below:

```
<html>
<head>
<title>the webSurvent Training Demo</title>
</head>
<body  bgcolor="white" text="black" />
<center />
<h2>Welcome to the webSurvent Training Demo<br />
</h2>

<form name="cfmclogin" method="post" action="/cgi-
bin/cfmccgi/websrv81.cgi">

<!-- System Specific (Probably Won't Change) -->
<input type="hidden" name="CFMC" value="/cfmc/test8.1/"/>
<input type="hidden" name="CFMCCFG" value="/cfmc/test8.1/ipcfiles/"/>
<!-- Tmpl locations -->
```

```
<input type="hidden" name="STUDY_DIR"
value="/cfmc/test8.1/websurv/studies/demo/"/>

<input type="hidden" name="TMPLS_DIR"
value="/cfmc/test8.1/websurv/tmpl/WS_default/"/>

<!-- Other Study Specific items -->

<input type="hidden" name="STUDYCODE" value="demo"/>

<input type="hidden" name="USE_PASSWDS" value="NO"/>

<input type="hidden" name="MAILTO" value="support@cfmc.com"/>



    <div>

      <fieldset>

                       <legend>Survey Login</legend>

      <label for="name">Name:</label>


      <!-- Use this instead of "Name:" line if everyone's named
"Respondent":

 -->

          <input type="hidden" name="name" value="Respondent">

<!-- -->

      <input type="text" name="name" id="name" tabindex="1" value="" />

      <br />

      <label for="password">Password:</label>

      <input type="text" name="password" id="password" tabindex="2" />

      <br />

      If you do not enter a password, one will automatically be assigned
for you.

      <br />

      <span><input class="submit" type="submit" name="start" value="Start"
tabindex="3" /></span>

      </fieldset>

    </div>



 <input type="hidden" name="USER_DATA" value="96.232.213.180">

</form>


<p /><hr />If you have any questions, or problems, please contact CfMC
Support by e-mail at
```

```
<a href="mailto:support@cfmc.com?subject=demo
indexnp">support@cfmc.com</a>.


<script src="/cfmcweb/js/initial81.js" type="text/javascript"></script>

<script src="/cfmcweb/js/user_settings81.js"
type="text/javascript"></script>


</body>

</html>
```

NOTE: For information on the JavaScript settings used above, refer to *Appendix H (JavaScripts)*.

Server down hidden input message

You can specify a "no_server" hidden input in your index page and give the URL of another page to be displayed when the cgi cannot connect to the server.

Example:

> '<input type=hidden name="no_server"
> value="http://www.acme.com/no_server.html">'

It allows you put up a more generic error message for webSurvent studies. And, it will match the language of the message to the language specified in the index.html page.

Index file requirements

First, you need a form called cfmclogin that is posted to CfMC's Web command processor (websrv81.cgi located in your Web server's cgi-bin/cfmccfg directory). Use CfMC's WEBUTIL or WCATUTIL utility to set up your HTML files; they will automatically include this and the other default variables and will prompt you to enter the values for others it needs.

CfMC system variables

CFMC – Identical to your $CFMC environment variable, the location of your cfmc software directory

CFMCCFG – Identical to your $CFMCCFG environment variable, the location of CfMC's communication files (${CFMC}ipcfiles/)

Template variables

STUDY_DIR – The primary location of study template files, usually you compile your study
TMPLS_DIR - The default location for template files if not study specific. It is usually either /websurv/tmpl/WS_default or /websurv/tmpl/skins/skin# if you are using a predefined skin.

Other study-specific variables

STUDYCODE – The name of the study

USE_PASSWDS – YES for closed study, NO for open, CATI or webCATI

Name and password variables

WebSurvent will format response boxes on the index.html page for respondents to enter name and password for both open and closed studies.

For **closed** studies:

NAME: Respondents enter their pre-designated name.

If a name is not important for the study, add ‘type=“hidden”’ and ‘value=“respondent”’ to the input definition. In this case, your raw sample file should have “respondent” in the name field (71.20) for every respondent.

PASSWORD: The password respondents will enter to start or resume an interview. It must match the pre-designated password in the sample file.

For **open** studies:

NAME: Respondents will enter their name variable.

PASSWORD: There will not be a pre-designated password for an open study. The variable may be automatically generated or respondents may enter their own passwords. To have the password automatically generated, do not include a password input variable. To have respondents enter their own password, include a password input variable.

Language variable

Also, language may be set with variables at the index page for webSurvent and webCATI. You can now set the language to be used in the upcoming survey by filling the variable LANGUAGE with the proper two-character code for that language. You can either have an index page for each language where this is set in a hidden field or you could prompt for the language and use a JavaScript to fill in the appropriate language code. For example, for Spanish it would be:

```
<INPUT type="hidden" NAME="LANGUAGE"  VALUE="SP">
```

This will cause the language in the questionnaire to be set to Spanish.

User_data variable

WebSurvent will pass the string contained in the hidden input USER_DATA into the survey. This hidden input is automatically written on every page of the survey if it is defined. USER_DATA can only have up to 120 characters.

The JavaScript function parse_query reads the strings and puts it into the data. Please refer to Appendix B for more details on how to do this. This function is only usable on the first page of the survey.

JavaScript controls and variables

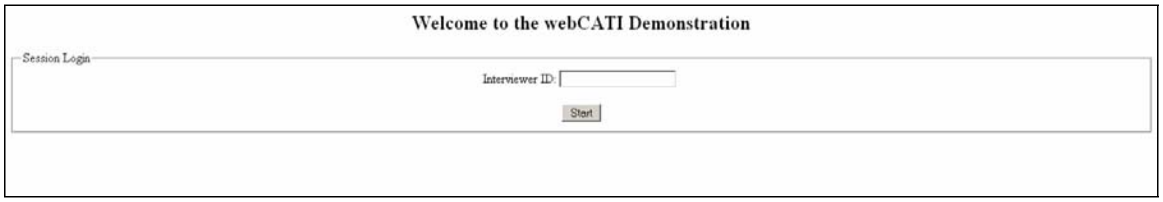
Please see the initial81.js file and other JavaScripts in *Appendix H* of this manual.

The webCATI index.html file

The HTML file index.html created by WCATUTIL is usually the first screen that the interviewers see when they access a webCATI questionnaire. You may reformat it using HTML code.

The startup file can be given any name you want. However, it is recommended that you use index.html because this is the default html file that is brought up when one accesses a URL. For example, if you type in http://www.cfmc.com in your browser, this loads the file "index.html" in the Web server www.cfmc.com's home directory. This would be the same as typing http://www.cfmc.com/index.html. Give this a try at your Web site.

Below is an example webCATI Index page as viewed through a browser:



NOTE: The value of CATI for the USE_PASSWDS input puts the Survent session into webCATI mode, and once in webCATI mode, Survent knows to access the employee.xxx file.

Required webCATI variables

A form called cfmclogin must be included in the index.html file for your webCATI study. If you use the WCATUTIL utility to set up your HTML files, it automatically includes some of the necessary variables and it prompts you to enter the others. For this reason, we strongly recommend that you use WCATUTIL to set up the HTML files for your webCATI studies.

System variables

CFMC – the location of your CfMC software directory

CFMCCFG – the location of CfMC's communication files (${CFMC}ipcfiles/)

USE_PASSWDS – must be set to CATI for webCATI mode

Template variables

STUDY_DIR – The primary location of study template files, usually your study directory

TMPLS_DIR – The secondary (default) location for template files (usually, {CFMC}websurv/tmpl/WC_default/)

Other study-specific variables

STUDYCODE – The name of the study, as named in your qpx study header (three to eight alphanumeric characters long, starting with a letter)

Interviewer ID variable

WebCATI will format a response box on the index.html page for interviewers to enter their Interviewer ID (as stored in the employee.xxx). The interviewer will need to enter their Interviewer ID in order to gain access to a survey.

Using WEBUTIL or WCATUTIL to generate index files

WEBUTIL and WCATUTIL collect information and create the index.html and supporting HTML files for your study. Once you have created the default index.html file for your studies, it is strongly recommended that you use WEBUTIL or WCATUTIL to update the default index.html file for each Web study, as it prompts you to input all the variables needed.

To run WEBUTIL,

Type:

webutil

The following prompts appear:

First, enter your study code or "q" to quit ==>

Closed studies require passwords, open studies do not.

Does ctest81 require passwords or not? (y/n) ==>

Enter the study's title (please, NO COMMAS)

NOTE: The study title will appear as '<study title> study' ==>

Enter the Study Contact name, this will appear on index.html and bannrbtm.tmpl ==>

Enter your Study Contact e-mail address, this will appear on index.html and bannrbtm.tmpl ==>

Enter domain name or press <Enter> for (default Domain Name) ==>

The home directory for html documents is /www/htmldocs. Enter the subdirectory from this location where ctest81's html file(s) will exist or press Enter for ctest81 ==>

These are covered in more detail in *Chapter 6.*

Here are available “looks” or “skins”:

Skin1: Simplest look. Uses only an initial JavaScript for entry into the survey

Skin2: Uses all JavaScripts provided. Some inline style code used for formatting.

Skin3: Basic stylesheet: Uses all JavaScripts and simple page formatting stylesheet.

Skin4: Medium Stylesheet: Uses all JavaScriptdscripts. Uses stylesheets for page and question formatting.

Enter corresponding number of skin or press Enter for default (WS_default) ==>

Copy index.html and supporting .html files to web area? ==>

Do you want to be informed by email when there are no more respondents after study has been deactivated? ==>

Here are the items filled in when using WCATUTIL:

Study Code – study name as assigned in the questionnaire header

Study Title – description that will appear on the index page as "Welcome to <study title>" and in the upper left-hand corner of the browser window

Name – contact name

E-mail address – contact e-mail address

Location of customized tmpl files – usually ${CFMC}websurv/studies/<studycode>

Default location of tmpl files – usually ${CFMC}websurv/tmpl/WC_default/

Html document directory for study – the sub-directory off of the html home directory where you want your study's html files to be stored (usually same as study name)

WEBUTIL and WCATUTIL create a local file called websurv.var that contains all of the variable definitions that must exist in the index.html file in order for webSurvent to function. The websurv.var file is just a log of the initial settings used. It's a quick way to view your settings.

WEBUTIL creates "retired.html" in addition to index.html. When the study has been completed, replace the index.html file with retired.html. If respondents attempt to log on to the study, they will see a screen with a message that reads, "This questionnaire has been discontinued."

WEBUTIL creates a MOVEHTML file so you can move your HTML files from the study directory to the Web area if you decide to edit them locally and move them later. To use it, type /movehtml".

Additional WCATUTIL command line options

WCATUTIL supports the following command line options:

USAGE: /${CFMC}/go/wcatutil <studycode> <contact="name,emailaddress">

... and any of the following options (defaults shown)

<title="study title" (def:"<study> Study")>

<domain=domainname (def: linsup.cfmc.com)>

<studydir=studydirectoryname (def:/cfmc/test8.1/websurv/tmpl/WC_default/)>

<wwwdir=studyhtmldirectoryname (def:<study>)
NOTE: Main html document directory "/var/www/html/" is implied for option above

<skin="skinname" (def:WC_default)>

<notify=y/n (def:n)>

<batch> - will read local wcatinit file

You can also utilize the editable wcatinit file located in the $CFMC directory. The user can then run "wcatutil batch".

CHAPTER 6

CfMC TEMPLATE FILES

"Template" files are files that are read by CfMC's Web programs to help create the Web pages, start the survey, or stop the survey. The page heading and buttons at the bottom of the page are examples of items controlled by templates.

The defaults for these templates exist in the ${CFMC}websurv/tmpl/WS_default (webSurvent) or ${CFMC}websurv/tmpl/WS_default (webCATI) directory. They may be left as they are, or they can be modified. After they are modified, the template files may be stored either in the study directory (the STUDY_DIR variable specified in index.html) to override the defaults, or you can modify the TMPLS_DIR variable to point to some other directory for the template files. File stored in the study directory take precedence over those stored in the default directories.

If CfMC Web programs do not find all the required template files in either the study directory or the template directory, they generate an error message. The template directory also contains several optional template files that are used to control the look of the pages.

Required templates

complete.tmpl - Thanks respondents and tells them the interview is completed and they may return to their browser.

suspend.tmpl – Tells respondents that the interview has been suspended and they may come back later to finish it.

terminate.tmpl – Tells respondents that the interview has been terminated.

error.tmpl – Tells respondents that an error has occurred and displays error numbers and text.

Page formatting templates

header.tmpl – This creates the html head tag for a webSurvent job. It is placed before the bannrtop.tmpl. Anything that would normally go inside an html head tag, such as the title tag, style sheet code, meta tags, etc., should be included in this file.

bannrtop.tmpl – This is a banner that appears on the top of every page - usually used to set the page font, background, etc. This can also provide text or an image at the top of each page.

btnsbtm.tmpl –This formats movement buttons that appear below the question text and answer list.

btnstop.tmpl – Executed after the bannrtop.tmpl and before the question. It formats movement buttons that appear above the question text and answer list. This is optional.

bannrbtm.tmpl – This causes a graphic image or text to appear at the bottom of every page or a JavaScript designation to be executed for every question. (For some useful JavaScripts created by CfMC, refer to *Appendix H*.)

Alternate formatting templates

In order to provide the user more control over the page webSurvent outputs to a browser, you may use alternates templates other than those described above. If you use the above .tmpl files, then the <html> and <head> tags are created entirely by webSurvent. If you would like to have control over these tags, you can use the following tmpl files. Make sure that you write an <html> and a <head> tag into them as some browsers may be problematic if they are missing.

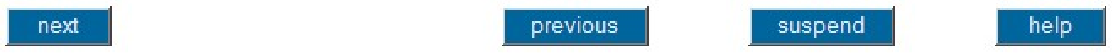
pagetop.tmpl – Replaces header.tmpl and bannrtop.tmpl. Make sure it contains a <html> tag, a <head> tag, and a <body> tag. In general, calls to style sheets and JavaScripts should be placed inside the <head> and </head> tags.

buttons.tmpl – Replaces btnsbtm.tmpl

pagebtm.tmpl – Replaces bannrbtm.tmpl.

Behavior: By default, webSurvent will use header.tmpl, bannrtop.tmpl, btnsbtm.tmpl and bannrbtm.tmpl if they exist. If you wish to use the alternate formatting templates, make sure you remove these from the directory. If btnstop or btnsbtm are specified, that template is in use. If both are specified, you will get two sets of buttons. If webSurvent cannot find either template, the browser will insert gray default buttons.

Controlling the display of the movement buttons



You can control which buttons are displayed at the top or bottom of a question using the {!ALLOW…} keywords, or by editing the file btnsbtm.tmpl or btnstop.tmpl. To change the position of the keywords, edit btnsbtm.tmpl. By default, the program displays buttons for "Next," "Previous," "Suspend" and "Help." To remove these with the {!ALLOW…} keywords, follow the chart below: Remove the following keywords with the following commands:

Movement button to remove	Command
Next	Always allowed
Previous	{!-ALLOW_BACKUP}
Suspend	{!-ALLOW_SUSPEND}
Help	{!-ALLOW_SPECIAL}

NOTE: In order for this to work properly, the <INPUT> tag of each button must have the NAME= defined on the same line.

WebCATI <body> statements

The default body statement for all webCATI screens (except in situations where error.tmpl is used) is:

```
<BODY TEXT="#000000" BGCOLOR="#FFFFFF">
```

The above <body> statement says text of the survey pages is to be displayed in black, and the background color of the page is to be white.

This will be used if there is no body statement in header.tmpl.

A body statement in header.tmpl. (or the default body statement if no body exists in header.tmpl.) will be in effect for the complete template and the questionnaire. It will NOT be in effect for error.tmpl.

A body statement in the questionnaire will override the body statement in the—header.tmpl.

A body statement in the error template will not conflict with any other body statements.

Template files for webCATI

The webCATI interview uses template files to:

Display pages between interviews

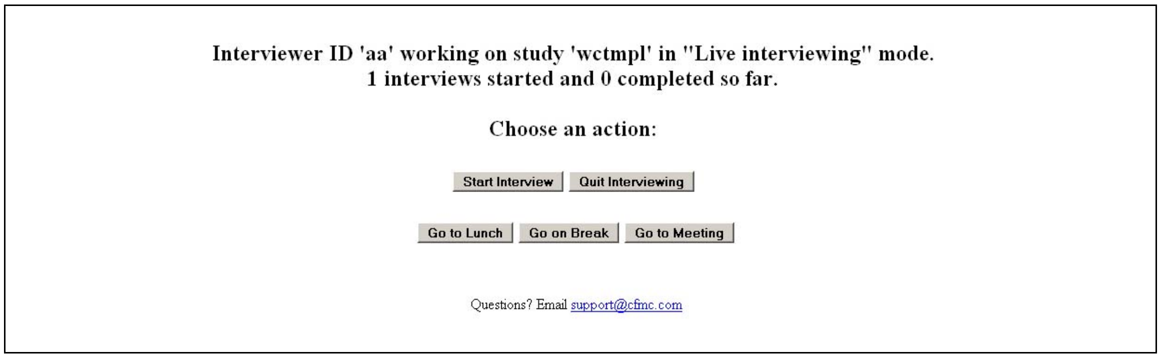
Control the top and bottom of the page for style reasons

Control the buttons that display on the page

The template files are located in $CFMC/websurv/tmpl/WC_default. And, they are:

qprompt.tmpl	The screen that displays between interviews
pagetop.tmpl	Top of each webCATI page, includes calls to JavaScripts
buttons.tmpl	Has the buttons at the bottom of each webCATI page
pagebtm.tmpl	Bottom of each webCATI page, includes custom JavaScripts
error.tmpl	Page displayed in case of a “blow” error

You can edit each of these files. If you’d like, you can include a custom set of these in the STUDY_DIR area (and point the STUDY_DIR variable there for those webCATI logins) and those will be used instead of the ones in WC_default.


Interviewer ID 'aa' working on study 'wctmpl' in "Live interviewing" mode.
1 interviews started and 0 completed so far.

Choose an action:

Questions? Email support@cfmc.com

qprompt.tmpl – This has various information about the study set by using system "defines" as well as the buttons to go into the interview, quit, go to lunch/break, etc. (note that the "lunch" and "break" screens are currently not editable). Here is the standard qprompt.tmpl:

```
>-define @statusbar  "to turn off the status bar for this page if used in the survey
<center>
<h2><br>Interviewer ID '@name'
>if @booth > 0
at booth '@booth'
>endif
working on study '@studycode'
>ifdefine @practice_mode
  in "Practice Interviewing" mode.
>else
>ifdefine @debug_mode
  in "Debug" mode.
>else
  in "Live interviewing" mode.
>endif
>endif
>ifdefine @caseid
<br>
Last case id completed was @caseid~.
<br>
>endif
```

```
<br>
@starts interviews started and @completes completed so far.
<br><br>
Choose an action: <br><br>

<INPUT TYPE="submit" NAME="Start.x" VALUE="Start Interview" style="font-size:14px;font-weight:bold">
<INPUT TYPE="submit" NAME="Quit.x" VALUE="Quit Interviewing" style="font-size:14px;font-weight:bold"> <br><br>
<INPUT TYPE="submit" NAME="Lunch.x" VALUE="Go to Lunch" style="font-size:14px;font-weight:bold">
<INPUT TYPE="submit" NAME="Break.x" VALUE="Go on Break" style="font-size:14px;font-weight:bold">
<INPUT TYPE="submit" NAME="Break.x" VALUE="Go to Meeting" style="font-size:14px;font-weight:bold">

>ifdef @debug_mode
<br><br>
<INPUT TYPE="text"   NAME="nextcard" VALUE="" size=25>
<INPUT TYPE="submit" NAME="Next" style="fontsize:14px;fontweight:bold" VALUE="Enter">
>endif
</h2>
```

Notice that you can easily add colors or change the font sizes. Note that if the interviewer is “Debug” mode, they will get an extra box displayed that allows them to type keywords such as “view” to view interviews or other commands.

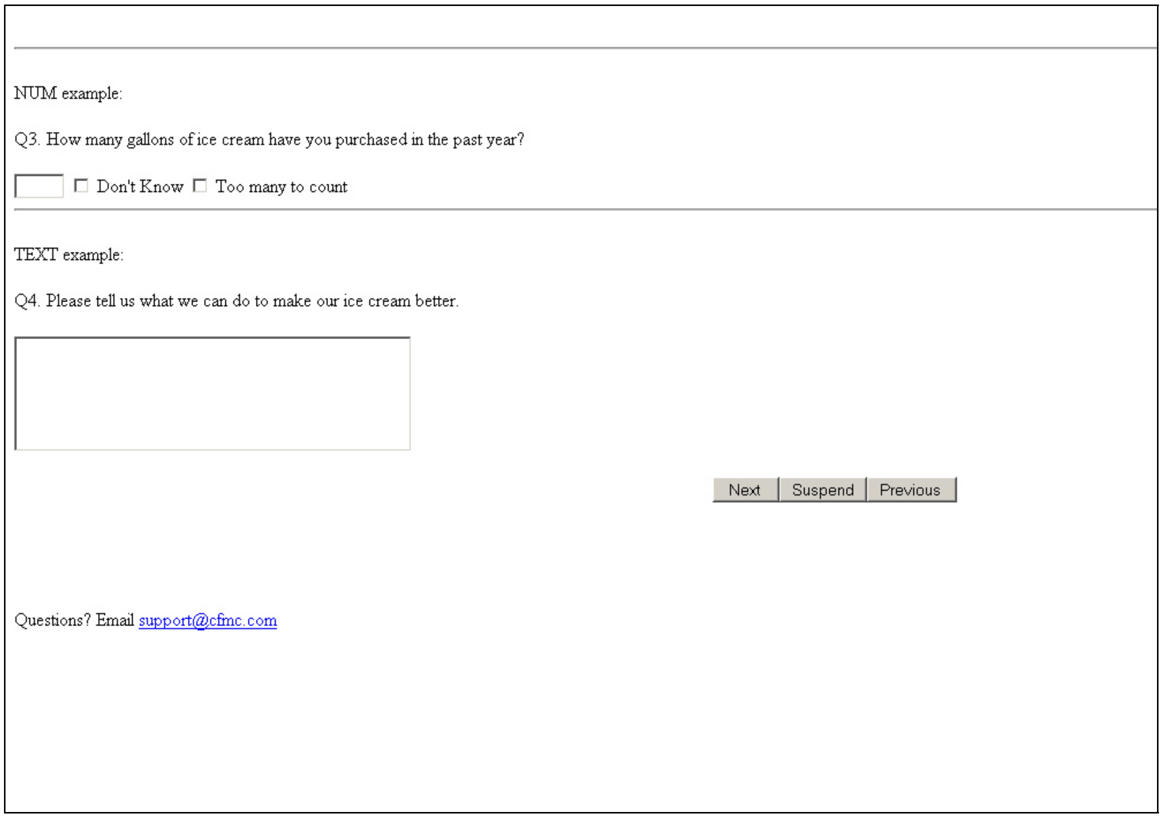
pagetop.tmpl – This file can include style elements and calls to our standard JavaScripts for webCATI:

```
<html>
<head>
<script type="text/javascript" src="/cfmcweb/js/cfmc_ws81.js"></script>
<script type="text/javascript" src="/cfmcweb/js/cfmc_tmpl81.js"></script>
<script type="text/javascript" src="/cfmcweb/js/user_settings81.js"></script>
<script type="text/javascript" src="/cfmcweb/js/placefocus.js"></script>
```

<title>WebCATI Survey </title>

</head>

Notice that there is a call to “placefocus.js”. This causes the focus to go on the first box on the screen so that the interviewer doesn’t need to use a Tab or the mouse. The other calls are to our standard JavaScripts. *See Appendix H*, the JavaScript documentation, for more information.



NUM example:

Q3. How many gallons of ice cream have you purchased in the past year?

☐ Don't Know ☐ Too many to count

TEXT example:

Q4. Please tell us what we can do to make our ice cream better.

Questions? Email support@cfmc.com

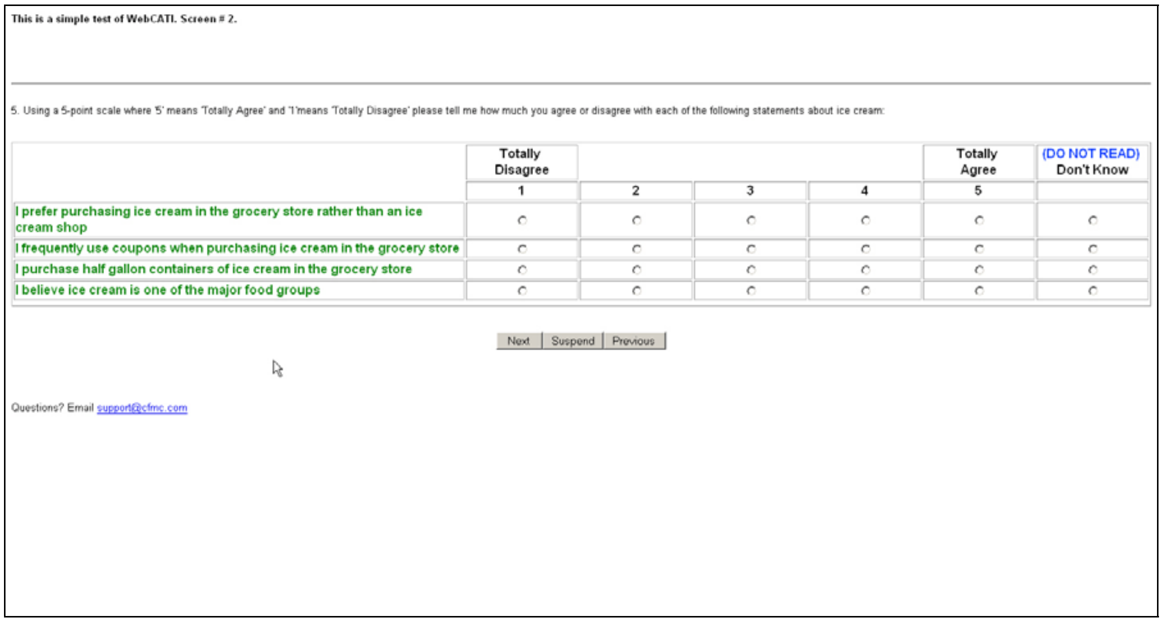
buttons.tmpl – This file includes the buttons you want to put on the page for the interviewer:

```
<!-- grey navigator buttons -->
<BR><BR>
<TABLE BORDER=0 ALIGN=LEFT CELLPADDING=0 CELLSPACING=0>
<TR>
<TD> <INPUT TYPE="submit" NAME="Next.x"  ID="Next.x"  VALUE="  Next   ">
>ifdef @suspend_allowed
<TD> <INPUT TYPE="submit" NAME="Suspend.x" ID="Suspend.x"   VALUE=" Suspend ">
>endif
>ifdef @backup_allowed
```

```
<TD> <INPUT TYPE="submit" NAME="Previous.x" ID="Previous.x"  VALUE="Previous ">
>endif
<!--
>ifdef @terminate_allowed
<TD> <INPUT TYPE="submit" NAME="Terminate.x" ID="Terminate.x" VALUE="Terminate">
>endif
>ifdef @special_allowed
<TD> <INPUT TYPE="submit" NAME="Special.x" ID="Special.x"   VALUE=" Special ">
>endif
>ifdef @promptable
<TD> <INPUT TYPE="button" NAME="Prompt"   VALUE="Prompt">
        <INPUT TYPE="hidden" NAME="nextcard"   VALUE="---">
>endif
-->
</TR></TABLE>
<BR CLEAR=ALL><BR>
```

The standard buttons that are used are “Next”, “Previous” and “Suspend”.

Notice that the “terminate”, “special/help” and “command prompt” buttons are currently commented out, if you want to use those you need to remove the “<!—“ and “-->” that surrounds them.



This is a simple test of WebCATI. Screen # 2.

5. Using a 5-point scale where '5' means 'Totally Agree' and '1'means 'Totally Disagree' please tell me how much you agree or disagree with each of the following statements about ice cream:

	Totally Disagree				Totally Agree	(DO NOT READ) Don't Know
	1	2	3	4	5	
I prefer purchasing ice cream in the grocery store rather than an ice cream shop	○	○	○	○	○	○
I frequently use coupons when purchasing ice cream in the grocery store	○	○	○	○	○	○
I purchase half gallon containers of ice cream in the grocery store	○	○	○	○	○	○
I believe ice cream is one of the major food groups	○	○	○	○	○	○

Questions? Email support@cfmc.com

pagebtm.tmpl – This has any custom JavaScripts and includes the “status bar” which tells the interviewer how far along in the survey they have come. You can also include or not include an email address for the interviewer to contact in case of problems:

```
<br />
<div>
<div>
<!--style="display: none">-->
<p>Questions? Email <a
href="mailto:@mailto?subject=question,study:@studycode,username:@name,password:@password"
>@mailto</a></p>
<div id="statbar" name="statbar"></div>
</div>
</div>

<form name="statbar">
<input type="hidden" name="statbar_perc" value="@statusbar~">
</form>
```

To use the “status bar,” you must have a “!status_bar” statement in the questionnaire.

error.tmpl – This tells the interview a “blow” error has occurred and displays the text of the error if there is any:

```
<center>
<br>
<h1> An Error Has Occurred! Please contact your supervisor and report:
<br>
<br>
<span style="background-color="color">@error_text </span>
</h1>
```

Defining a look using “skins” in webSurvent

CfMC’s Web programs provide a way to create predefined looks that can be used for any jobs you choose. In addition to the tmpl files in WS_default, CfMC also provides four pre-defined skins. These skins simply use different default tmpl and html files to control the look and the wording. When running WebUtil to set up a job, you are given the option to choose which skin you wish to use or the defaults. You have the choice of adding looks and skins that are relevant to your site.

The skins CfMC provides are:

Qs_look1: The quickSurvent Look: This gives you the look and feel of the standard quickSurvent look. This uses all js scripts. This look also uses stylesheets for page and question formatting. Represents the first standard "default" qS lookand feel.

skin1: This gives you the simplest look. This uses only an initial JavaScript for entry into the survey. There's no style sheet and minimal formatting.

skin2: This uses all JavaScripts provided. No style sheets are used for formatting.

skin3: This is a basic style sheet. It uses all JavaScripts provided and style sheets for simple page formatting.

skin4: This is a medium style sheet: This uses all .js scripts. It uses style sheets for page and question formatting

skin9: This is a Light blue and white skin, used by the CfMC Service Bureau. It allows for a more complex style sheet: This uses all .js scripts. It uses style sheets for page and question formatting

Creating your own webSurvent looks with skins

The following files are used by skins:

Template files

Bannrtop.tmpl – controls the top of each page presented in webSurvent.

Bannrbtm.tmpl – controls the bottom of each page.

Header.tmpl – The contents of the <head> tag. It can be used to present the <title> tag on each survey page. It is in this file that your calls to external js and css files should be made as well.

Btnsbtm.tmpl – controls the presentation and functionality of each of the buttons on-screen during a webSurvent survey.

Suspend.tmpl – This presents text and links to respondents when a survey is suspended.

Terminat and **complete.tmpl** – terminat is shown when an !spc,b is hit and complete is shown when a survey is completed.

Error.tmpl – presents any number of errors a respondent may get while accessing the survey – including a bad username/password combination.

Where files are stored

There are several directories that are important in the skins feature:

/$CFMC/websurv/tmpl/skins – in this directory, there is a subdirectory for each skin on the server. All 8 template files listed above are required in these subdirectories. This is the default location for all tmpls used for that look.

/$CFMC/websurv/archive/skins – again, there is a subdirectory for each skin on the server here. These files are used when running webUTIL. The following files are in the subdirectory for a skin:

> Skin#.qpx – This is not required, but this provides an easy way for programmers to grab a default shell for any given skin.
>
> Description.txt – The text in this file is presented when running webUTIL.
>
> Tmpl directory – requires header.tmpl – This is the only file that has to be in your study directory (/$CFMC/websurv/studies/studydir) because it has study-specific text in it. webUTIL changes the appropriate text and copies it into the study directory.
>
> Html directory – contains indexp.html (for closed studies), indexnp.html (for open studies), retired.html and help.html. Each html file contains specific variables that will be filled in by webUTIL (such as studycode, survey title, and server name (if applicable)).

/www/htmldocs/cfmcweb/ - In 8.1, this directory stores any JavaScripts, style sheets and images that are accessed by the looks we provide. You can place your own files there as well, but be careful to not overwrite any files provided by CfMC.

CSS – This will store CfMC external cascading style sheet documents that are called in to Web surveys.

JS – houses CfMC JavaScript files called in to Web surveys.

Images – folder for CfMC images called in to Web surveys.

Creating skins

Make a new subdirectory in the following directories:

> /$CFMC/websurv/tmpl/skins/
>
> /$CFMC/websurv/archive/skins/

In /$CFMC/websurv/tmpl/skins/ there will be the default templates for that particular skin. Bannrtop.tmpl, bannrbtm.tmpl, and error.tmpl may need to be modified for the specific look of the skin. The remaining tmpls do not necessarily need to be made specific for the look of the skin.

In /$CFMC/websurv/archive/skins/, place description.txt and skin#.qpx. Two additional subdirectories also must be created here:

html – this stores indexp.html, indexnp.html, help.html and retired.html. All must be modified the specific skin look.

tmpl – this should house header.tmpl – the only template that will be unique from study-to-study.

If you need to use other skin-specific external css files and/or JavaScripts, these files may be placed in their appropriate directories in /www/htmldocs/cfmcweb/ or in your own directories in the Web-accessible area of your server. Permissions should be set to world readable or the files will be inaccessible from a browser.

Setting up the skins files used by WebUtil

This applies only to the files placed in /$CFMC/websurv/archive/skins.

When running WEBUTIL, certain files are changed, based on your input. In order for this to work, placeholders must be used in these files, and they cannot be removed. So, you have to make sure that they exist.

The applicable files are indexp.html, indexnp.html, retired.html and header.tmpl.

Placeholders are always in upper-case, and they begin and end with an underscore (_). The placeholders used are:

TITLE: Study title entered in webutil (used in all html files and header.tmpl)

WEBVARS: Hidden variables used in the index page.

MAILTO: Contact name and e-mail address

STUDYCODE: Used as subject in e-mail link.

Controlling .tmpl files from the questionnaire using !html_defines

To define variables to control the tmpl files from your questionnaire, you can use the command {!html_define}. These commands behave like >define in the standard software, but they only can be referenced in the template files.

You can use html_defines to fill in tags or the information in the tmpl files to pass information to the tmpl files. Or, you can combine them with >if_define to create conditional displays in the tmpl files.

In your questionnaire spec, use {!html_define <name> <text>} to set the command you wish to use. Note: !Html_defines may only be set outside of a !grid statement.

In the tmpl files, use @name to do text replacement or ">if_define @name <thing to do> >endif " to create conditional pages. If another character immediately follows the @defined name, use a ~ (tilde) to separate the name from the rest.

Here are some specific examples:

Define the body color you wish to use in your spec:

```
{!html_define body_color ffffff}
```

Reference @body_color in the body tag and when the page is sent to a browser. @body_color will be replaced with what is defined in your spec. You could use this to change the body color of your webSurvent pages at any time.

```
<body bgcolor="@body_color~" text="000000">
```

You could show buttons only on certain pages by using:

```
{!html_define new_button}
```

in the specification and

```
>if_define @new_button
```

```
<input type="button" name="description"
style="background:#a4c17c;color:#d9ffa4" value=" Description ">

>endif
```

The Description button would only show up on pages where new_button has been defined. To turn off the define, simply use {!-html_define new_button}.

You could pass data from the survey into the end template files such as suspend, complete and terminate.

In the questionnaire spec create an html_define that backreferences either a data location or the answer array.

```
{!html_define q2_code \|q2|}

{!html_define q2_text \:q2}
```

In the tmpl files any reference to @q2_code or @q2_text would be filled in with the value from the survey.

CHAPTER 7

HOW TO OPERATE THE CFMC STUDY SERVER

Getting the questionnaire up and running

WebSurvent and webCATI use the standard CfMC study server. Use the following steps to get your questionnaire up and running on the Internet.

1. Start the CfMC Server. To start the CfMC study server in the background, type:

```
server bg ←
```

This will first build or clean out old entries from the server's configuration files, then run a background study server and log all events running under the server.

2. Run the Survent Supervisor to make sure the server is running by typing:

```
super ←
```

If Supervisor starts, this indicates the server is running properly. To see if a particular study is loaded, type SUPERVISOR command:

```
dis ←
```

To see if a study will load, type "qss <study>".

3. Set up the suspend timeout:

If you wish to have a timeout enacted after a certain number of minutes of inactivity on a study, you can change the setting by putting a command in the file ${CFMC}control/parmfile. This will be set for all studies:

```
suspend_timeout:## (## is the number of minutes)
```

If you want to specify a different timeout for some study, from the supervisor type:

```
server:suspend_timeout:<study>##
```

NOTE: If you have an open study, display the name and password as soon as possible so that if the respondent suspends or gets bumped off, he or she can come back in and resume. *See 9.4 Starting an Interview.*

Starting an interview

To start an interview, activate a Web browser, such as Netscape or Internet Explorer. Then type your server's Web address on the browser's "go to" line. For example:

http://www.yourcompanyhere.com/<complete path to study directory>

The browser's window should display the index.html file for that study. (See the Troubleshooting section in this manual if you cannot access the index file)

Suspending and resuming an interview

Respondents may suspend the interview whenever they desire, unless {!-allow_suspend} to disallow it has been defined for the section of the interview they are answering questions for. To resume the interview, they must return to the original URL, enter their name and password and they will resume where they left off. Questions within {!suspend} and {!resume} blocks will be executed just as they would in a normal CATI interview.

Auto Suspend

WebSurvent sessions that have had no activity for a time span greater than that defined in the suspend_timeout option will be automatically suspended. When this occurs, the suspend block of your study will be executed except for any questions that need input from the respondent. This is a feature that allows you to save the interview data even if the respondent just "goes away", so that when they come back they can resume where they left off. This works in conjunction with "auto resume" below.

Auto Resume

After "autosuspending" an interview, if the respondent keys an entry and submits the page they were on, this will cause Survent to "auto resume" from that point on and the survey continues. Submitted question screens after a "suspendall" command has been issued from the supervisor will behave the same way. Unfortunately, they will have to re-submit the last answer that they had sent. This is why it's a good idea to set the suspend_timeout to a reasonably high value, especially if you expect a lot of open-end information. Be very careful about your use of the "suspendall" option as well.

Fast Resume

"Fast Resuming" was created to handle situations where a respondent's browser session has been lost because the respondent has closed the browser, has operating system difficulties, the machine loses power and most other communication breakdowns. If any of these events happen, the respondent will be allowed to re-enter the interview (at the index.html point) and then resume at the point where he or she left off.

Monitoring a Web interview from a terminal mode supervisor

Using this mode, the monitoring process does not interpret html; it just passes the code on to the monitor screen. Therefore, the monitor screen will be difficult to read. A practical usage of monitoring is to see if the interview is still active. To monitor a webSurvent or webCATI interview from the Supervisor, do the following:

To find the webSurvent ldev numbers, type:

```
dai all ← (display all interviewers)
```

Each respondent will be listed as a number from 2803 on.

To monitor the respondent of your choice, type:

```
        mon #<ldev> ←
```

Press **<ctrl>-y** to get out of monitor mode.

In Supervisor, if you want to see what webSurvent stations are running, you can use the "DAI_WEBSURVENT" or DAIWS" command.

Enter:

```
    daiws ←
```

The format looks like this:

```
    LDEV    Study       password    PID    last        qffname
    10004: rvsurvey     NOMCBI      20087  10:54:52    rvsurvey.qff
```

"LDEV" is the CfMC-assigned device number, "Study" is the study name, "password" is the password on that sample record, "PID" is the Survent process ID for that session, "last" is the last time the session contacted the server, and "qffname" is the questionnaire file name.

If there are no webSurvent sessions running, it will tell you so.

If you enter "daiws <study>" it will just show the stations for that particular study.

In addition, there is a "DAI_WEBCATI" or "DAIWC" command that you can use to display active interviewers in WebCATI.

Enter:

```
    Daiwc ←
```

The format looks like this:

```
    LDEV    Study       PID          last        qffname
    10004: rvsurvey     20087        10:54:52    rvsurvey.qff
```

This displays the same information as DAIWS, except that there is no password. And, if there are no webCATI sessions, it will say so.

If you say "daiwc <study>" it will show only the stations for that particular study.

> **NOTE:** See the Supervisor Manual or the *Survent Technical Manual, "Chapter 4: Conducting the Interview*," for more details on how to use the Supervisor. See the appendix for monitoring webCATI using a browser.

Deactivating/retiring a study

To shut down an individual study temporarily while the study server is still running:

Enter a SUPERVISOR command:

```
deactivate <studycode> ←

        or

retire <studycode> ←
```

These commands disallow new respondents from entering the study. When respondents attempt to access a deactivated survey, a message from the error template informs them that the study is temporarily unavailable.

Current active interviews are allowed to finish. Once the last interview has finished, the study files will be unloaded from the study server, thus allowing any file changes or replacements to take place.

After deactivating a study, you may observe that some interviews don't go away. Another command to use is:

```
server:suspend_timeout ←
```

Sometimes, you can simply use this command and all interviews will be inactive. However, you are more likely to find that some of the interviews are suspended and that some are not, possibly because they are abandoned sessons. If you need to get into the study immediately or if you are too impatient to wait for the remaining open sessions, you can use the following command to force them off the server:

```
server:suspendall:<study>←
```

If you are still left with interviews that are hung up, you can use the following command:

```
server:defunct ←
```

This will remove session information for sessions that do not have an active Survent running. For more information on the use of this command, please refer to *Step 4 of Shutting down the Study Server in Chapter 10* of this manual.

Reactivating a study

To make the study active again: Enter the SUPERVISOR command:

```
activate <studycode>  ←
```

Deactivation notification

Once the study has been deactivated, it may take a while for current interviews to finish. It may be tedious to run SUPER and issue the “dis” command every few minutes to see when these people have finished or at least have suspended. A system has been setup so that the project manager can be e-mailed when the last session has finished after a “deactivate” command is issued for a study. As you may recall, this is a function of WebUtil.

If there is a file <studyname>.sh in the ${CFMCCFG} area, the study server will attempt to execute that file once the last person for the study has left webSurvent and the study has been unloaded. The shell file can be any type of shell script. To send e-mail, the script can look something like this:

```
!/bin/csh
set study = test
set pmemail = "<your email address here>"
set body = "Study $study now unloaded from $CFMC on $HOST"
set subject = "Study $study now unloaded"
echo "$body" | mail -s "$subject" $pmemail
exit
```

Using the cgi_hitlog file to display information on interviews

The file cgi_hitlog.<ldev of study server> exists in the directory ${CFMC}super/. This file shows the course of webSurvent interviews, including when people log into and out of webSurvent. It displays information in the following format:

```
Time - ? – study:passwd:name:IP:browser info
```

For example:

```
18Jan2001 10:57:05 – I – gnum2:HVBMPZ:Bob:123.123.123.123:Mozilla/4.0
(compatible; MSIE 5.0; Windows 98; DigExt)
```

The following are things that you may see in your CGI hitlog:

Time – date and time respondent accessed the study (18Jan2004 10:57:05)

Stage/Status of the interview:

A – Previously completed; can't do survey

I – Study inactive

! – Phone record up-in-the-air and no corresponding Survent session can be found to connect to

B – Beginning interview (between intro.tmpl and 1st question) – interview begun

C – Completed interview (between last question and complete.tmpl)

H – Tried to access a hidden fone record

E – Error accessing fone record (no such sample record)

F – Study retired

N – Can't open study

O – Over Limit (Hitting maximum capacity for concurrent users at a given time, based on software license. In this case, a respondent will see a message saying the server is currently busy and to try again later.)

P – Can't load study for some reason

R – Resumed interview (between last question answered and complete.tmpl)

S – Suspended interview (between last question answered and suspend.tmpl)

T – Terminated, aborted interview

W – Interview resulted in a blow error

Z - Zombie Survent process cleared (eg. with server:defunct)

Study – study code (gnum2)

Password – Respondent’s password (HVBMPZ)

User – Respondent’s submitted name (Bob)

IP – Respondent’s IP address (123.123.123.123)

Browser info – Info about the Respondent’s Browser (Mozilla/4.0)

NOTE regarding status “!”: This status reflects a situation where the respondent’s Survent session blew or the study server crashed and left the phone record with no status.

The remedy for this is to shut down the study, run Foneutil and use the V)erify U) Return on-the-floor numbers and X)Fix/re-link options for the fone file.

Shutting down and starting up a server

Shutting down a study

Before shutting down the server, you need to shut down each study. To shut down a study, first check to see if any interviews are running:

Enter a SUPERVISOR command→: Dis ←

If there are interviews running, wait until they are completed if possible, then deactivate the study:

Enter a SUPERVISOR command→: Dis ←

To shut down a study even though there are interviews still running, issue the Supervisor command:

Enter a SUPERVISOR command→: server:suspendall:<studycode> ←

NOTE: Use this command carefully. It forces the study server to immediately suspend all of the active Survent processes for the specified study. Although information from incomplete interviews is saved in resume files, some respondents’ interviews are cut short before they have been completed.

Repeat Step 1 to make sure no interviews are running.

Shutting down the CfMC study server

First, check to see if any interviews are running under the server by issuing a “dis” command:

Enter a SUPERVISOR command: dis ←

Deactivate any studies that have interviews running and wait until the interviews clear. To deactivate each active study, do the following:

Enter a SUPERVISOR command: deactivate <studycode> ←

If it is necessary to shut the server down while some interviews are still running:

Enter a SUPERVISOR command: server: suspendall ←

NOTE: Use the suspendall command with caution. It forces the study server to suspend every active Survent process immediately. Although information from incomplete interviews is saved in resume files, some respondents' interviews are cut short before they have been completed.

Repeat Step 1 to make sure no interviews are running.

NOTE: If you have any sessions that are hung up, you will see them in the Supervisor.

If there are hung sessions (sessions showing on the list when you know they are not active), do the following:

Enter a SUPERVISOR command: server: defunct ←

This will clear the session that's hung up.

Shut down the study server:

Enter a SUPERVISOR command: server: down ←

Bringing up the server, reactivating studies

If the server is/was shut down and you want to start it up again:

Enter command: server bg ←

Start the Supervisor.

Enter command: super ←

As soon as the server is up, studies will be reactivated.

CHAPTER 8

SPECIAL FEATURES OF WEBCATI

Optional Web-based interviewer authorization

The cgis and html pages in this directory are designed to allow you to authorize booths (This is synonymous with “extension” in call-center mode.) on particular studies, and for interviewers in those booths to log in to those studies. Supervisors can authorize a booth or a range of booths on a particular study from the "wcstart.shtml" page.

Interviewers log in with their booth number at "wclogin.html" and that booth number is used to determine which study to start on. The interviewer is prevented from starting on a study unless they are authorized on that study.

Since interviewers don't log on via a “standard-style” index page, the settings for each study are located in a "<studyname>.wc" file. The settings in these files are combined with the interviewer ID, etc, collected by the "wclogin.html" form, which is then submitted to the standard CFMC cgi (also listed in the "<studyname>.wc" file), and interviewing procedes as normal.

For each study you wish to make available, copy the "example.wc" file into the "WCAUTH_INFO" directory as "<studyname>.wc" and modify the settings to match what you would normally put in your normal index.html log-in page. Add a WEBMON_DIR if appropriate.

Authorizing Interviewers

Point your browser at the "wcstart.shtml", wherever you installed it. Enter a range of booths. Select a study from those shown (only studies which have a "WCAUTH_INFO/<studyname>.wc" file are shown). Click the “Start” button. The newly started booths should be added to the table at the bottom of the page.

If there are problems, such as no studies showing up in the “Active Jobs” box, check the Apache error_log – most likely the permissions aren't right on the “wcstart.cgi”, or server-side includes aren't configured properly in your webserver.

Ranges of booths can be as complicated as "100-120,124,50-60,150,155". If you want, a booth can be started on multiple studies at once, allowing the interviewers to select which study to interview with.

To clear booths, simply type the range you want to clear. Then, click the “Clear” button.

This is not real-time management; it doesn't stop interviewers who are currently interviewing, nor does it change interviewers from one job to the next. In order to use a real-time management function, you need to use SURVSUPR. It may be possible that this can be integrated with survsupr in the future. This allows you to authorize stations on various studies only at the beginning of a shift.

In addition to the booth number, interviewers are required to enter a “passcode.” This passcode is basically a check to stop interviewers from entering the wrong booth number.

To enable this feature, you must create a file named "booth.passcd" that has one line per booth, each line with “booth passcode.”

After that file is created, you need to add to the "wclogin.html" file the following input:

```
<INPUT TYPE="text" NAME="passcd" id="passcd" VALUE="" size=8>
```

and to the "wcpopwin.html" file, add the line

```
<input type="hidden" name="passcd"   id="passcd"  value="">
```

Interviewers log in

Interviewers log in by pointing their browsers at the "wclogin.html" page. They will be prompted for the booth number and interviewer ID. Currently, the booth number is the dialer extension number at that booth.

Multiple environments

In some instances, it may be useful to set up different areas for managing different studies, different study servers, etc. The easiest way to do this is to run through the installation again, using different values for “WCAUTH_PATH” and “WCAUTH_INFO”.

Multiple machines

It is possible to have the wcauth stuff installed on one machine and have the studies it controls live on different machines. All you need to do is give the full URL to cgi on the remote machine as the "POST" value in the "study.wc" file.

Using a dialer with webCATI

Call center mode with a dialer

Running webCATI with a dialer in call center mode means that the interviewers are located in a call center and connected to the dialer.

When in call center mode: webCATI interacts with any dialer CfMC currently supports – MSG, EIS, SER or Stratasoft. No dialer software upgrade is required.

The Parmfile DIALER: line should be the same as it is for Survent. That is, there is no difference for a call-center-webCATI parmfile dialer line, vs.a Survent dialer line.

Input fields should be added to the index.html file to specify Interviewer ID and extension.

For example:

```
Interviewer ID: <INPUT type="text"      NAME="NAME"     VALUE="bb">
Extension: <INPUT type="text" NAME="EXTN"     VALUE="30">
```

In call center mode, the interviewer goes to the URL/index.html and enters the interviewer ID and extension # of the booth (the study name is in a hidden field). Upon submitting this information, the interviewer is presented the first question of the questionnaire. The extension may be provided as a hidden field in an index page set up for that specific booth.

Remote-access mode

Running webCATI in remote mode with a dialer means that the interviewer is located somewhere other than the call center (e.g., working from home) and connected to the dialer.

When utilizing the remote access feature, there are a few additional considerations:

> **NOTE:** At this time, the remote access feature is available only through a special version of the Marketing Systems Group (MSG) dialer called Remote Interviewing Module (RIM). It may work with other dialers in the future.

An MSG dialer software upgrade is needed in order to utilize the Remote Interviewing Module (RIM).

HTML <INPUT> fields need to be specified in the index.html file for Interviewer ID, extension and remote phone number.

For example:

```
    <Interviewer ID: <INPUT type="text" NAME="NAME"     VALUE="bb">
Extension:
    <INPUT type="text" NAME="EXTN"     VALUE="30">
Remote Phone Number:
    <INPUT type="text" NAME="REM_PHONE" VALUE="4157770470">
```

A DIALER input field can be specified instead of an EXTN field, and the stdysrvr will then choose an unused extension on that dialer. That line would look something like this:

```
    <INPUT type="hidden" NAME="DIALER" VALUE="1">.
    (where value= the dialer number as specified in your parmfile)
```

Either the DIALER field *or* the EXTN field is specified, but not both. The Parmfile should have an additional DIALER: line.

Remote mode needs a separate dialer##: line from the call center dialer## line. So your parmfile should have two dialer statements (possibly more depending on your shop's setup), something like:

```
    dialer1 for normal cati stations/ call center mode webCATI stations
    dialer2 for remote interviewers
```

Each dialer line must have a disjointed set of extensions.

In remote interviewing mode, each interviewer uses a browser to access the Web survey site over the Internet/Intranet. The study name is in a hidden field. The interviewer then enters *either* a dialer number (in which case the stdysrvr chooses the next available extension from that dialer's extension list in the parmfile) *or* their extension, plus a phone number or an alpha name; when using an alpha name (e.g. 'home', 'homeofab', etc.), the employee file stores the phone number to be used for the dialer's connection to the interviewer's location.

The webCATI study server then sends the dialer the interviewer phone number and the extension to be associated with it (along with a VIVR_STARTED message) so that the study server can correlate the phone number with the appropriate campaign.

The dialer dials the number, waits for the interviewer to answer the call, at which point the dialer notifies the study server that the extension is connected (with a VIVR_CONNECT message), is ready to receive calls, and adds the extension to its extension list.

As the dialer encounters live respondents, the calls are automatically connected to the lines for each remote interviewer that is available in the same manner as a typical call center. Once connected to the dialer the interviewer has continuous interviewing, meaning the interviewer need not log back into the study on the index.html following each interview. When the interviewer completes a call, the study server notifies the dialer that the line is now available for the next call.

Aside from the VIVR_STARTED and VIVR_CONNECT messages, the dialer interaction is exactly the same for webCATI as for terminal-mode CATI.

WebCATI parmfile options

You may modify the following options in the ${CFMC}control/parmfile file.

Parmfile settings apply to all studies running under the server.

If you are using a dialer, set the dialer line in the parmfile by stating (depending on the number of dialers you are running):

DIALER:

This tells SURVENT the type of dialer you are using. The syntax is:

```
DIALER: <EIS/EIS,DUMMY/GERRY>
```

If you plan to allow SURVENT to give special statuses to numbers that have been returned from the dialer with a disconnect status, you need to specify "EIS,DUMMY". This mode allows you to set up your own “statusing” rules by having a station signed on the study with interviewer type 9, which would receive the numbers after they are returned from the dialer to “re-status” them before allowing interviewers to talk to them.

```
DIALER##:
```

This tells the program specific information about the dialer or dialers you are using. You may have up to 20 dialers, and you need one DIALER:## line for each to tell the program where the dialer is and what ports and extensions are involved. The syntax is:

```
DIALER##: <IP address>,<read/write sockets>,<extension #s>,<SOUND>
```

Here is an example:

```
DIALER01: SOCKET=192.183.2.2,5000,5001 41-100
```

The 01 is the dialer number. 192.183.2.2 is the IP address for the dialer on your network (or you could use its NAME, such as “DIALERX”). 5000 and 5001 are the read/write sockets for the dialer machine (you *must* use 5000/5001, 5002/5003, etc. pairs with EIS dialers). 41 – 100 are the CfMC dialer extension numbers you are assigning to the phones connected to the dialer. These must be in a sequential range where 41 is the first physical dialer extension and 100 is the last.

(Refer to *Chapter 6* of your *Survent manual,* for additional dialer options and information.)

After_Blow options

AFTER_BLOW: <option>

Suspend -- with the suspend option, when a survey gets a blow error, Survent attempts to back up one question and do a suspend. Before attempting the suspend, the interviewer is told "Tell your supervisor ... " If the suspend is successful, you are told so, and a suspend record will be created and placed in the .r_ subdirectory; nothing should be put into the .b_ subdirectory (unless the suspend fails). The interviewer session is then stopped.

Blow – (default) The default is to "BLOW"; that is, stop the interviewer session, save the blow record in the blow (.b_) directory, and print the error message.

Continue – The continue option makes the suspend file, and then continues interviewing. This allows interviewers to resume suspended records once the cause of the "BLOW" has been found and fixed. Be careful using the "CONTINUE" option, because interviewers may continue interviewing and never report the problem.

```
EMPLOYEE_ID: ##.##
```

This will accept an alternate location in employee file for login information, where ##.## is the alternate location. A login ID of one to four characters will be checked against columns 1-4 of the employee file, while an ID of 5 characters or greater will be checked against the alternate field. For example, if the interviewer enters CHAR as their ID, the program will look at columns 1-4 of the employee file; if the interviewer enters CHARLES as their ID, the program will look at the alternate columns.

You cannot use or overlap 41-44 because those are already used by CfMC, but you can go out to 300.

```
SOCKETS: YES
```

Sets up communications between webCATI and the CfMC server using TCP/IP protocol (sockets). This option should have been added to the parmfile when WSSETUP was run during the webCATI software installation.

Browser-based monitoring using webCATI

Configuring webCATI Survent for Web-based monitoring causes Survent to write files containing the current page of questions and the last set of answers, some information about the interviewer and some information about the current interview. These can be referred to as “monitoring pages.” The answers are from the last page presented because the software cannot get the answers for the current page until the interviewer submits the page (when Survent gets the answers to the current page, but sends the next page of questions to the browser.)

There are four files included in the installation package to be used for the monitoring process; those files are called monitor.php, monpane.php, monlib.inc and monstyle.css. The “monitor.php” file sets up a frame for viewing the information contained in the monitoring pages, and allows you to select which interviewer to monitor. The “monpane.php” file displays the information in the sub-panes. The “monlib.inc” is a php file that contains functions used by the php pages, and contains all of the Web monitoring configuration settings. The “monstyle.css” style sheet allows you to customize the look of the various parts of the monitoring page.

The Web-based monitoring runs using PHP pages inside of the Web server. Consult your Web server's documentation for how to install PHP if it isn't installed already.

For instructions on installing this feature, please consult CfMC’s “System Administrator's Guide: Software Installation and Setup (for Linux).”

Setting up monitoring

Add a parmfile setting “htmldocs: DOCUMENT_ROOT”, where “DOCUMENT_ROOT” is the Web server document-root directory.to the file $CFMC/control/parmfile. Restart your CfMC server so that the parmfile change takes effect (usually, by saying “server:down” in the supervisor, then “server” at command prompt).

For all studies you want to monitor, you'll have to add another hidden field to the index page: “<input type=hidden name=WEBMON_DIR value=MONITOR_PATH>”. Start a webCATI interview.

Open a browser window and point your browser at “your.machine.com/cfmcweb/webmonitor/monitor.html”

On the monitoring screen, choose which interviewer you would like to monitor from the drop-down box in the upper left-hand side of the screen (Now Monitoring).

On the right-hand side of the monitoring screen, the question(s) the interviewer is currently seeing will be displayed. On the left-hand side of the screen, the responses to the previous page of questions will be displayed (Monitoring/Extension).

Remove the WEBMON_DIR variable from the index.html file to disallow monitoring for that study.

“View” mode in webCATI

To 'view' interviews you can now point your browser to "//cfmcweb/view". It will prompt you for a study name and ID and then the usual “view mode” options. (*See more on the use of VIEW mode in Survent, Chapter 4, section 4.1.5*)

If you log on with a "debug" mode, ID it will allow you to alter responses.

If you specify an "A" in the employee.xxx file columns 42-45 that interviewer can use View ALTER mode and make changes to the data. If you are in View "Alter" mode, you wwon’t be required to save the changes to each page. When you are done with a case, the program will prompt you as to whether you want to save the changes made to that case.

Also, you can enter a label in the “Label to GOTO” box to move from question to question in addition to using the NEXT/PREVIOUS buttons.

APPENDIX A

TROUBLESHOOTING

If you are having trouble with a webCATI or webSurvent study, the following steps may help determine where the issue lies:

> **NOTE:** Many, if not all, of the steps can be skipped if you have a specific symptom. For instance, if you see a blow message in the browser, then you know that the webserver, cgi, and studyserver are all running just fine, and that it's a questionnaire problem.

Verify that the Web server is running.

Using your browser, point at any page on the server (e.g., http://your.website.com).

> If the page loads, then the Web server is running.
>
> If you get any error in the 400s, then the Web server is running, but you've probably asked for a page that isn't on the machine.
>
> If you can't load the page, then your Web server is not responding and may need to be restarted; speak with your Systems Administrator.

Check that the websrv81.cgi program runs correctly.

Change directories into the cgi-bin directory (this may look something like: "cd /var/www/cgi-bin").

Run websrv81.cgi from the command line by typing "websrv81.cgi"

> You should see the message "Fatal Error: CONTENT_TYPE must be set" if the program is running correctly. The error occurs because the .cgi is expecting to be run by the Web server.
>
> If you do not see the "Fatal Error" message, call CfMC Support. There are a number of reasons why the program may not be running correctly; a couple of examples are: the file is not there at all, or the file does not have execute permission.

Make sure your studyserver is running.

> If the studyserver is running but the study is not, one of the study's files may not be where the studyserver is looking for it.

Try running the study in regular terminal mode Survent.

Try running a generic test study.

Create a spec (e.g., test.qpx) with a few questions in it (maybe a field, a num, and a var question).

Compile the spec and use "loadstudy" to copy study files to their appropriate directories (e.g., loadstudy test).

Create an index page using WCATUTIL.

Access the study through your browser (e.g., http://your.machine.com/websurv/test/). This can help determine if there's a configuration problem or just a problem with your study.

Survent Issues

Test the study server:

Make sure there is a sockets statement in the parmfile by typing:

```
grep sockets ${CFMC}control/parmfile ←
```

This should return:

```
sockets:yes
```

If not, add the sockets:yes line to the parmfile.

Start the study server by typing:

```
server ←
```

This will display server run information on the screen.

Start another session. Then start the Supervisor by typing:

```
super ←
```

Test the study server by typing several Survent Supervisor commands.

NOTE: See the Chapter 4 of the *SURVENT Technical Manual* for more information about Supervisor commands.

Test Survent under the study server

Compile demo.qpx in ${CFMC}websurv/studies/demo directory by typing:

```
mentor demo.qpx –demo.lfl
```

Use the loadstudy demo command to load the study to the interviewing area.

Start networked Survent by typing:

```
netsurv ←
```

Enter demo for the questionnaire QFF file name.

Enter dbug for an interviewer ID.

You should be placed on the first question of the questionnaire.

Enter all responses or type abort to exit.

Web server issues

Test the httpd process

Check to see that the httpd daemon is running by typing:

```
ps auxw | grep httpd (For Linux)
ps –ef | grep httpd (For other types of UNIX)
```

If httpd is not running, start it. (Consult your System Administrator if necessary.)

Test the Web server

To determine if the Web server is functioning and serving up Web pages, use the files' date.html and date to test the server.

Copy the file ${CFMC}websurv/archive/date/date.html to your Web server's standard HTML documents directory (usually htmldocs).

Copy the file ${CFMC}websurv/archive/date/date.cgi to your Web server's cgi-bin directory. (The location of the cgi-bin directory is defined in the NSCA access.conf file and is usually cgi-bin in the Web area.)

Try to access date.html with your browser. Use the Web address:

```
http://<holstname or ipaddress>/date.htm
```

It should return a date in the following format:

```
Mon June 14 09:02:20 PST 2004
```

If there is no response at all, then there may be a problem with httpd.

If the message Error 404: File not found appears, check the location of the date file.

If the message Error 403: Forbidden appears, check the permissions on the date file.

If you can access date.html but the server cannot find the date.cgi file, it probably means that the cgi files are not installed in their proper places on the system.

Determine where the cgi files should be installed on your system and install them in the correct location. Also determine if any restrictions exist on your system regarding the execution of cgi files.

If you receive ERROR 500, it is because the Web server cannot complete the request that the browser has made. There is only one error that is CfMC's: "premature end of script headers." This usually means that there is a problem in the cgi, resulting from junk data being delivered by the browser.

What to do: Try and access the server from a different browser or from outside the firewall.

WebSurvent Error Messages

The following is a list of possible webSurvent error messages.

Errors numbered **9100-9149** usually indicate an error from the study server or with communication between the study server and cgi.

Errors numbered **9150-9199** usually indicate an error with cgi internally.

If you have any problems that you can not resolve, call the CfMC Technical Support Hotline at (415) 777-2922.

Study server, communication, cgi and logfile errors

ERROR: Server error!

Explanation: The server encountered an internal error and was unable to complete your request.

What to do: If this is consistantly reproducable at the client site, CfMC may be able to try to diagnose and fix the reporting of this problem.

ERROR: Premature end of script headers: websrv81.cgi

Explanation: Apache (and other web servers) put out this mesage when a cgi cgi quits early. Why the cgi quit early is the real question in this case, and usually that's very hard to find out. Often it's due to a communications problem between the cgi and the survent process that it's talking to, but there are other condidtions that could lead to this message.

What to do: If you think this is a server error, please contact the webmaster. If this is consistantly reproducable at the client site then CfMC may have ways of trying to diagnose and fix the reporting of these problems.

ERROR: 500

Explanation: This means that the web server itself encountered an error which prevented it from fulfilling the request of the browser. This is internal to the web server and is outside of our control.

What to do: Check the Apache error logs as to why this might be occurring.
Sometimes just trying to send the page again will resolve the problem.

ERROR: 9100: Bad length for CGI_FONEINFO structure

Explanation: The websrv81.cgi and the CfMC study server are not of the same version. The message between the CfMC study server and websrv81.cgi has been truncated. Your websrv81.cgi probably was not updated when your study server was.

What to do: Check to make sure that websrv81.cgi in your Web server's cgi-bin directory is the same size and date as the cgi file in your ${CFMC}go/ directory. If it is not, copy the file from ${CFMC}go to your cgi-bin directory and make sure it has execute capability for all.

ERROR: 9101: No study named (<studycode>) available

Explanation: Either the study does not exist at all or the individual study files (qff, quo, fone, tr) do not exist or load with errors. Check for specific problems. Possible errors in the log file could be:

> The qff, quota or fone data directories aren't readable/writable by the server.
>
> The .qff, .quo, .fon files are not readable/writable by server.
>
> The .qff, .quo, .fon files do not exist in their appropriate directories.
>
> .qff files: ${CFMCQFL} (default: ${CFMC}qff/)
>
> .quo files: ${CFMCQUOTA} (default: ${CFMC}quota/)
>
> .fon files: ${CFMCFONE} (default: ${CFMC}fone)

The TR file can't be created in the data directory. Either the data directory (${CFMCDATA} default: ${CFMC}data/) is not writable or the user who is running the study server cannot write to the TR file.

What to do: Check the possible explanations above and act accordingly. Run the Supervisor and try to load the study using spi <studycode>. This will tell you what files, if any, are missing. The server does not have to be shut down for the changes to take place.

ERROR: 9102 This study (<studycode>) has been retired

Explanation: The Supervisor command retire <studycode> was used for this study. This should appear only in the logfile (${CFMC}super/ll######). A respondent will see the error.tmpl file unless the active index.html has been replaced by retired.html.

What to do: To make the study active again, use the Supervisor deactivate command, followed by the activate command.

ERROR: 9103 This study (<studycode>) is inactive

Explanation: The Supervisor command deactivate <studycode> has been used for this study. This should appear only in the logfile (${CFMC}super/ll######). A respondent will see the error.tmpl file.

What to do: To reactivate the study, use the Supervisor command activate <studycode>.

ERROR: 9104 This study (%s) cannot be started now. Please try again later.

What to do: Try to run NETSURV for the study and note what happens when you try to start it. Send the logfile (${CFMC}super/ll######) that includes the error plus any background information about what was happening on your system when the error occurred to CfMC Support.

ERROR: 9108 The name/password combination you entered [(name)pass)] is invalid. Please try again.

Explanation: This message should only occur for closed studies. Either the requested phone record cannot be found because it's hidden or the requested phone record cannot be found because it's not in the file.

What to do: Find the name and password that the respondent was trying to use. Try to find this record in the study's phone file.

ERROR: 9109: Unable to create new record for name/password

Explanation: This message should only occur for open studies. It means that a new phone record cannot be added to the phone file because it would be a duplicate of an existing name and password. The fact that the phone record already exists should have been caught before this point.

What to do: Check the condition of the FONE FILE (run FONEUTIL). Check the logfile (${CFMC}super/ll######) for this name/password combination. A more descriptive error might be found near where the message occurs.

ERROR: 9110 Unable to locate resume record for name/password [(name)pass)] in study [(<studycode>)]

Explanation: The resume record for this interview has been deleted from ${CFMCRESUME}<studycode>.r_/. The definition of ${CFMCRESUME} (default: ${CFMC}resume/) has changed since the interview was suspended. The study name has changed since the interview was suspended and the resume file was not moved to the new resume directory.

What to do: Check for any of the above occurrences. If you can find the old RESUME file, copy it to its correct directory.

ERROR: 9111 can't locate record [%s:%s] for study [%s]

Explanation: Record cannot be found in the fone file.

What to do: Check the fone file to see if the record exists.

ERROR: 9120 no auditing process (snapper) or maximum Survent processes

Explanation: The CfMC snapshot auditing program (snap81), which should be running in the background, is not running, or your operating system has reached its process limit.

What to do: If the snap81 program is not running on your system, run the CfMC program SNAPPIT (${CFMC}go/snappit) to bring it up again. If this fails, deactivate your studies and bring your CfMC study server down, then bring it up again. If your operating system has reached it's process limit, either increase the number of process allowed by your UNIX operating system kernel (check with your system manager) or, decrease "HTML_LIMIT" in ${CFMC}control/parmfile.

ERROR: 9121: Unknown error talking to Survent for study [(<studycode>)]

Explanation: The study server received an unknown error from Survent. Websrv81.cgi has received this because the study server has been bypassed.

What to do: Send the logfile (${CFMC}super/ll######) that includes this error plus any background information about what was occurring on your system when the error occurred to CfMC Support.

ERROR: 9122 You appear to be already in the study.

Explanation: The number is up-in-the-air. This is probably due to a Survent blow error or a timeout in phase 3.

What to do: Deactivate the study. When there are no longer any active sessions, go into the fone directory and run foneutil. When prompted for a studyname, enter the name of the study you are working on. Run option V, then option X. This will allow the respondent to re-enter the survey. You will want to check in the log files to see if this was caused by a blow error, and if so, correct the error.

ERROR: 9150 Cannot connect to server at port (port) ldev (ldev)!

Explanation: CGI cannot communicate with the study server. There can be many reasons

for this:

The CFMCCFG variable (default ${CFMC}ipcfiles/) has not been set correctly in index.html

The owner of the cgi process (the http daemon) does not have read access to the stations file in /cfmc/cfg (The /cfmc/cfg directory is a symbolic link)

What to do: Take a look at the source of your index.html and find the following line:

```
<INPUT TYPE="HIDDEN" NAME="CFMCCFG" VALUE="/cfmc/ipcfiles/">
```

Compare the location defined between the quotes of the “VALUE” definition to the $CFMCCFG environment variable that your study server is using. You may check your CFMC environment variable by signing on as the same user that you sign on to when you run your study server and then typing “echo $CFMCCFG”. If nothing is returned, then the default will be the ipcfiles directory under your main cfmc software directory (i.e., ${CFMC}ipcfiles/). Make sure these definitions agree.

Make sure the /cfmc and the /cfmc/cfg directories and the stat##.cfg file have read access for the httpd user. Make sure it is world-readable if you’re not sure. The UNIX command to change access is “chmod”; type “man chmod” at your UNIX prompt for more info. If you make a change to the access of the file, you should shut down and restart your study server.

Make sure when you type “cd /cfmc/cfg” then type “pwd”, “/cfmc/cfg” is returned. Unfortunately, CfMC software cannot work well with symbolic links. Hard links are probably OK. Shut down the study server, remove the symbolic link for /cfmc/cfg, then make the /cfmc/cfg directory (with read/write access to the owner of the study server process and read for everyone else), then bring the study server back up.

ERROR: 9151 Unterminated <BODY> tag in QFF

Explanation: An HTML <body> statement has been found that does not have a closing “>”.

What to do: Check the questionnaire spec file for a <body> statement. Find the offending <body> statement, fix, recompile and reload the new .qff file only.

ERROR: 9152 Unterminated <_> tag in template: <template file name>

Explanation: An HTML tag (like <body>) has no ending (i.e., no “>”)

What to do: Check the specified template file for html tag errors.

ERROR: 9153 CGI received a bad message code

Explanation: Websrv81.cgi does not recognize the code received by it from the CfMC study server.

What to do: This may reflect a version conflict. Check your software install procedure. Check to make sure that websrv81.cgi in your Web server's cgi-bin directory has the same size and date as the cgi file in your ${CFMC}go/directory. If it does not, copy the file from ${CFMC}go to your cgi-bin directory and make sure it has execute capability for all.

ERROR: 9155 CGI does not know which Survent to talk to (no process_id)

Explanation: Websrv81.cgi does not have the process ID number that it needs in order to associate the response with the proper Survent process.

What to do: This is a symptom of a catastrophic situation. Shut down your system and inform your system manager. Send the logfile (${CFMC}super/ll######) that includes this error plus any background information about what was happening on your system when the error occurred to CfMC Support.

ERROR: 9156 Unable to locate template file: <tmpl file>

Explanation: The specified template file cannot be found or the specified template file cannot be opened.

What to do: Check the study's index.html for the STUDY_DIR definition and TMPLS_DIR to see if the specified template file exists in either of these directories. If the file exists, probably websrv81.cgi does not have read capabilities for the file or any directory in the file's path.

ERROR: 9157 Oversized template file: <tmpl file>

Explanation: The template file that cgi is trying to load is too big (>64kbytes or 32000 characters – roughly 32 screens)

What to do: Check out the specified file and reduce the size if it's too big.

ERROR: 9158 template_to_browser() (getcore()/opentest()/startread()) error

Explanation: There is not enough memory to load the template file or the template file is locked by another process.

What to do: This may be the sign of a catastrophic situation. Have your system manager check memory usage. Also check the defined template file for errors. Send the logfile (${CFMC}super/ll######) that includes this error plus any background information about what was occurring on your system when the error occurred to CfMC Support.

ERROR: 9159 "Name" must be at least (1) characters

Explanation: This comes up in both open and closed studies, when a respondent does not enter a name.

What to do: Make sure that you are giving the respondent the opportunity to enter a name in index.html. Then inform respondents that they must enter a name.

ERROR: 9160 “Password” must be at least (3) characters

Explanation: For an open study:

The respondent has been given the opportunity to enter his or her own password and the respondent has not entered a password or the password entered has less than the lower limit of 3 characters.

For a **closed** study:

The password defined in 51.10 of your phone file is too short or it was entered incorrectly by the respondent.

What to do: For open study:

Inform respondents that they must enter a password of more than 2 characters.

For **closed** study:

Check the password field (51.10) of your phone file and rebuild accordingly. Check the hitlog for incorrectly entered password. It’s likely a respondent error.

ERROR: 9162 Received a bad (unrecognized) message.

Explanation: The message type number received by websrv81.cgi from the CfMC study server is not recognized by websrv81.cgi. “#” is the message number. Probably your websrv81.cgi was not updated when your study server was.

What to do: Check to make sure that websrv81.cgi in your Web server’s cgi-bin directory has the same size and date as the cgi file in your ${CFMC}go/ directory. If it is not the same, copy the file from ${CFMC}go to your cgi-bin directory, and make sure it has execute capability for all.

ERROR: 9163 Communications timeout during phase (<record_lookup, record_create, survent_startup or interview>)

Explanation: There is a communication problem somewhere between the respondent’s browser and the respondent’s Survent process. Either the study server has crashed, was aborted or a runaway process has rendered the study server unusable.

What to do: Check your study server to see if it’s running and/or your study server’s log file (${CFMC}super/ll######) for errors. If the server is running, deactivate the studies and bring the server down. If the server is not running, check your study files for integrity, fix and verify your phone file, and try to bring the server back up.

WebSurvent Blow Messages

WebSurvent blow messages will appear with same text and for the same reasons as regular Survent blow messages.

Miscellaneous Errors

ERROR: 3853 Can’t find the file of shop parameters (name: /cfmc/control/parmfile)

Explanation: The http daemon does not have read access to ${CFMC}control/parmfile.

The get-around is to change the read/write access of ${CFMC}control/parmfile: (e.g., “chmod 744 ${CFMC}control/parmfile”)

Browser Errors

ERROR: FATAL ERROR: REQUEST_METHOD must be set to POST, not GET.

Explanation: This error occurs when there is a problem with the parser in the browser. This usually occurs in AOL versions 4.0 and 5.0. With 4.0, people cannot access the survey; with 5.0, you will get a prompt asking if you wish to continue. If a respondent answers “Yes” to the prompt, the respondent can continue into the survey

Similar problems have occurred with other browers, i.e., Compuserve 2000, but the problems have not been consistent enough to document.

What to do: The solution in all cases is to upgrade the browser or access the survey through a different browser.

Appendix B

INSTALLED WEB-RELATED DIRECTORIES

The ${CFMC}websurv directory

The ${CFMC}websurv directory is used to store CfMC standard files and your study files (although you may change this location if you'd like), as specified below:

archive/webutil/ - Examples of initial/log-in pages used for closed and open studies used by the WEBUTIL utility.

archive/images/ - Images that are used by the demo study included with

the webSurvent distribution. Move these to the images subdirectory off of your html document directory. (/home/httpd/html/images).

archive/javascript/ - JavaScript command files used by webSurvent.

archive/massmail/ - Example files used in conjunction with the MASS MAILER utility.

archive/tmpl/ - Archive of default template files used in the webSurvent process

studies/demo/ - The demo study files that comes with the webSurvent installation.

tmpl/ - Default template files that are used in the webSurvent process.

The HTML/Web area

The following default directories are used to store HTML-related files that will be used in your study.

Js – Where all the JavaScripts that are being used for the study reside.

Images – Where various images are kept. These would include buttons and other images.

Styles – Where any stylesheets that will be called in reside.

Appendix C

PROGRAMMING MULTIPLE LANGUAGES

CfMC supports character sets for all languages. In addition, CfMC supports language-specific error messages for the more popular languages. You may use more than one language in the same questionnaire. If you do, you use “\Lxx” as the lead-in character for a particular language. The interviewer in Survent or WebCati may switch languages at any time, while we recommend that a participant in webSurvent suspend the survey and return to the index to change language, or you can switch languages programmatically.

Choosing a language organization method

There are various ways to program languages using Survent/webSurvent. The first thing you need to decide is how to manage the languages:

If you have separate sample and quotas for each language, you would just build a separate study for each language. This is the easiest method.

If you have mixed sample and quotas, but will know the language of the respondent you are calling, you can write separate questionnaires that all talk to the same sample and quota files. This is also quite simple.

If you need to “combine” languages such that you can switch from one to another any time in the questionnaire, you will need to program all the languages into one questionnaire. This requires entering all the text for all languages into each question.

In the first two cases, because the questionnaires are separate, the only statement you need is a Language=(set=<charset> Character_set=(<language set>) <other options>) statement in the header of the questionnaire. Other than that, the languages are just typed in like English for the text and responses.

In the third case, you need to specify when you switch languages in the text and responses using \Lxx, where “xx” is the language, or [\l=xx] to start a response list section for that language.

If you use the UTF-8 character set, you can have all languages in the same questionnaire. If you use any of the other character sets, you will need a ***separate questionnaire*** for each character set. A character set can be any of UTF-8 (universal), Multi-Byte (Chinese/Korean), Shift-Jis (Japanese), or Extended_ASCII (European languages). So, if you are using non-UTF-8 character sets, you can put all the multi-byte languages into one questionnaire, Shift-Jis into another one and all the Extended_Ascii languages in another.

You can always have English with any character set: you can regard English as being “contained” in any of the sets. You can have as many languages as you want within any given character set. The separate questionnaires for each character set can use the same study name and share quota and phone files, writing to the same .tr file.

Using translation documents for language text

Typically, when writing multi-language questionnaires, the text is sent out to be translated in a file and a duplicate file is returned with the translated text. There is a way that you may reference translation documents for all your text instead of having to cut-and-paste the text into each language questionnaire. This can be used for any language script, but it is probably more trouble than help if you only have a few languages. The more languages, the more savings. You can send the same template out to be translated, and then reference the translated text across languages. Once this is set up, you can modify the original language documents and the changes will be included the next time you compile the script.

By maintaining separate documents for each language and employing the use of "include file" references, the programming of the spec can occur simultaneously or before the translation of the survey. More importantly, editing can be done to the documents without touching the questionnaire.

Saving from MicroSoft Word and Excel

Translations in MS Word format must be saved to text format to be compiled by mentor.

Let's see the necessary steps to save a Chinese Word document to Big5 or UTF-8 to be read in a text editor like Ultra-Edit.

The "save as" Word menu should present a list of possible encodings to choose from,

If you choose Big5 (one of the possible "Chinese traditional" encodings), then probably you will have to choose font MingLiu and Big5 encoding to display appropriately the ideograms in the text editor.

If you choose UTF-8 instead of Big5, usually there is no need to change font in the text editor. You can then use this text file as qpx and build the syntax around it or do a copy and paste job from the saved text file to the qpx (being careful not to break any ideograms, especially if you did not save to UTF-8). Alternatively, you could choose to use external language files and use the saved text file as one of the external language files.

Saving from an Excel document is similar to saving from a Word doc, although it can be slightly more convoluted. Let's assume you have a Japanese Excel document. Excel does not present a list of encodings for text files when you want to "save as" , so you choose Unicode text when saving, producing a Unicode tab delimited text file (no specification of actual encoding). If you open this file in Word, you can "save as" choosing the encoding you need: either Shift Jis or UTF-8.

Programming details

The languages that will be used must be specified in the header:

language=(character_set=<charset> set=(<lang letters>)

Example:

[xxxx language=(character_set=Extended_ascii set=(FR DE IT DU EN))]

There are also a few language keywords you can you use to fully control your multi-language spec.

The complete syntax for a language header keyword is (this is broken into two lines due to page margins):

```
LANGUAGE=( SET=(lang1,lang2,langn) CHARACTER_SET=   SPEAKING=langn
DEFAULT_LANGUAGE=langn or **  (-)CHECK_FOR_MISSING_LANGUAGES)
```

Only “set=” is required. The “Speaking=” parameter would be used if you want to set the starting and language; if you don't specify it, it defaults to the first language on the list.

The “Default_language” defaults to the first language on the “set=()” list, but you can change it at any time. This would be the language used if you didn't specify \L on some text. If you set default language to “**” it will display unmarked text in all languages.

“Check_for_missing_languages” makes sure all languages are specified on each question that has any language specified. The default is not to check. “\L**” accounts for all languages, unless there are other \Ls in the text, in which case there must be one for each language in the “set=”.

Here's an example:

```
        [study language=(set=(en,fr) check_for_missing_languages)]

{Q1:
\L**This is fine.
!disp}

{Q2:
\L**And so is this.\LenEnglish\LfrFrench.
!disp}

{Q3:
\L**But not this.\LenEnglish.
!disp}
```

Also, language may be set with variables at the index page for webSurvent and webCATI. You can set the language to be used in the upcoming survey by filling the html variable LANGUAGE with the proper two-character code for that language. You can either have an index page for each language where this is set in a hidden field or you could prompt for the language and use a JavaScript to fill in the appropriate language code. For example, for Spanish it would be:

```
<INPUT type="hidden" NAME="LANGUAGE"  VALUE="SP">
```

This will cause the language in the questionnaire to be set to Spanish.

“Include” file references

The syntax to reference a region of an include file is:

> Syntax:
>
> &<file> (‘start/’end)
>
> or
>
> {!Include <file>(‘start/’end)}

Since you may be using this often, it is less work to use “&<file>” references.

The ampersand must be the first character of a new line. Markers within the single parenthesis point to the beginning and end of a section of text in the include file. In the following example, the first include file contains only one section (‘b1/’e1).

> Example:

```
&multiby('b1/'e1)              "include file for multibyte
```

The possible character sets are “Extended_ascii”, “Shift-Jis”, “Multi-byte”, and “UTF-8”* (version 8.1+).

Setting up the Include files

The three examples are &extend, &multiby, and &shiftjis; for readability, the extended ASCII example will subsequently be referred to.

Each section of the text of the questionnaire must be accounted for – display text, error messages, text of questions, and response lists. Each place a \L<lang letters> command is used would be considered a section when creating the &filename.

Example:

```
{ SCR0:  10.1
Please indicate which main business unit of .... you primarily work with...
<br>
!FLD, , 1
1  Automobile manufacturer
2  Automobile manufacturer2
3  Automobile manufacturer3
4  Automobile manufacturer4
5  Automobile manufacturer5
}
```

From above, “Please indicate…..” and responses 1-5 are separate entries for the include file:

```
'scr1
Please indicate which main business unit of ... you primarily work with ...
'scrr1
1  Automobile manufacturer
2  Automobile manufacturer2
3  Automobile manufacturer3
4  Automobile manufacturer4
5  Automobile manufacturer5
'escrr1
```

…etc. The questionnaire spec, if using only the base language's (English in this example) include file, would look like:

```
{ SCR0:  10.1
&incen('scr1/'scrr1)
<br>
!FLD, , 1
[\Len]
&incen('scrr1/'escrr1)
}
```

This example gets translated into Dutch, French, German, Italian, Chinese and Japanese. This example has the translated files named 'inc' followed by their two-letter language code. This code will be shortened later, but the variable scr0: now looks like this:

```
{Scr0: 10.1
\LDU
&incdu('scr1/'scrr1)
\LFR
&incfr('scr1/'scrr1)
\LGE
&incge('scr1/'scrr1)
\LIT
&incit('scr1/'scrr1)
!fld
[\LDUen]
&incde('scrr1/'escrr1)
[\LFR]
&incfr('scrr1/'escrr1)
[\LGE]
&incge('scrr1/'escrr1)
[\LIT]
&incit('scrr1/'escrr1)
}
```

At this point, the repetitive code can be placed inside a ">repeat" structure.

```
>repeat $a= @lang
\L$a
 &inc$a('scr1/'scrr1)
>endrep
!fld
>repeat $a=@lang;
 [\L$a]
 &inc$a('scrr1/'escrr1)
>endrep
```

Making defines

The "@Lang" reference above is referred to as a ">define". Note: you need to retain consistency in the sequence of language codes in >repeat and >define statements. You can define anything that you don't want to have to retype later (this helps in keeping the code consistent):

```
>define @lang FR,GE,IT,DU,EN
```

Multibyte/Shift-jis character sets (Chinese/Korean/Japanese)

In order to display multibyte/Japanese characters, you need to use encoder software (Big 5, Njwin, Unionway) or an Operating System native to the language (for example, a proper Windows version for that language). Some text editors are able to show multibyte/Japanese characters choosing a particular font, such as Tahoma (Thai), Simsun (Chinese) or MS Mincho (Japanese).

If the files you are using are encoded to UTF-8, you should not need to use either encoder software or special fonts in the text editor.

Multibyte editing hints

For multibyte and shift_jis, there are problems editing text. The various multibyte/Japanese editors and operating systems have varying degrees of success at opening each other's files.

Original text should be saved as MS-DOS text in using Microsoft Word. It is recommended that you from MS Word as:

.txt (plain ASCII) .txt (Text Only with line breaks) .doc

Once again, you should be able to avoid most of those issue if you are operating in UTF-8.

Recommended editors

Epsilon, Emacs, Ultraedit, Textpad are commonly used editing software. Always have translators save any files as "text only" or "MS-DOS text" on the machine where the files were created. NEVER have it saved as unicode. It is difficult to edit the files on your machine; always send the translator changes. If there are CfMC spec changes (such as, things other than question text or response list text), make sure you are extremely careful when editing.

If your text appears as boxes or question marks in your text editor, the file is either not properly encoded or it has been transferred incorrectly from an OS other than the OS native to the language. If boxes or question marks appear in your Web browser, but appear correctly in your text editor, check your study header, meta tags and Apache configuration settings, as this means your browser or server is emulating the wrong character set.

Managing the message file

The msgfile is a central repository for the error messages. Any error message that is changed there will be shared by all the studies and processes in that environment. The msgfile is a binary file, so it cannot edited directly. We edit a text file called msgfile.raw instead and then we use the "makemsg" utility that reads msgfile.raw and builds msgfile. Version 7.7's msgfile.raw contains lines in extended ASCII, multi-byte and shift_jis encondings. Version 8.1's msgfile.raw is shipped in the same format as 7.7's but can be easily converted to UTF-8 as we provide the makemsg.pl utility to do so. The utility is called makemsg.pl. If you call it thus:

```
mkmsg.pl utf8
```

the program will read in the non UTF-8 file msgfile.raw and output a UTF-8 msgfile.raw and a UTF-8 binary msgfile, which you can then copy to ${CFMC}control. The UTF-8 msgfile.raw will have these lines:

```
zh0010 zh multibyte chinese_simplified
de0010 de extended_ascii german
dk0010 dk extended_ascii denmark
fr0010 fr extended_ascii french
it0010 it extended_ascii italian
pt0010 pt extended_ascii portguese
sp0010 sp extended_ascii spanish
```

```
sv0010 sv extended_ascii swedish
ct0010 ct multibyte chinese_traditional
ja0010 ja shift_jis japanese
ko0010 ko multibyte korean
0010   -- ascii standard_english
```

automatically converted to:

```
ct0010 ct utf8 chinese_traditional
de0010 de utf8 german
dk0010 dk utf8 denmark
fr0010 fr utf8 french
it0010 it utf8 italian
ja0010 ja utf8 japanese
ko0010 ko utf8 korean
pt0010 pt utf8 portugese
sp0010 sp utf8 spanish
sv0010 sv utf8 swedish
zh0010 zh utf8 chinese_simplified
0010   -- ascii standard_english
```

If you want to add new language codes you will need to add them in the section above. In general, we advise to run all existing 7.7 jobs in a 7.7 or 8.1 environment with a non UTF-8 msgfile. If you need to move a 7.7 job to a 8.1 UTF-8 environment, then it is much easier to convert the qpx and all language files (if any) to UTF-8 with the linux/unix utility *iconv* than to build a msgfile with UTF-8 and non UTF-8 error messages. An example of running iconv is:

```
iconv -f iso_8859-1 -t utf-8 -o utf8en.raw  langen.raw
```

The above will convert the iso_8859-1 file langen.raw to the UTF-8 file UTF8en.raw.

If you wish to create an all encompassing msgfile.raw with all encodings, you can use your editor of choice, mentor, perl or other programming language to build a msgfile.raw that contains messages in all encodings. You will find out however that you might run into a problem with the choice of the language codes if you want to stick to the "official" ones.

Creating your own error message

You may specifically control the text of CfMC messages for a particular questionnaire using the {!ERROR_MSG ####} compiler directive. This generally is placed at the top of the questionnaire.

Example:

```
{!ERROR_MSG 1512
fr1512 Vous devez entrer un texte
1512  You must enter some text    }
```

This will display “Vous devez entrer un texte” instead of the default “You must enter text” when that CfMC error message displays in “French” mode.

The structure of the error message must follow the structure of its counterpart in msgfile.raw including the %s %d, etc. The %<letter> can go anywhere within the message as long as there are an equal number from the original message. For example, 3 %<letter> in the original must have the 3 %<letter> in the new message.

The character limit for this compiler directive is 2000 total, inclusive of system variables such as %s and %d.

You can also edit the CfMC “msgfile” directly if you want these messages to be a permanent part of your system. First, you will need to edit the file “msgfile.raw” found in the CfMC Control directory. Copy it to another name and edit it. If you open it, you will notice a list of the languages there are error messages for. If your language is not on that list, select any unique two letters for your language.

Not all messages in the msgfile appear to interviewers/respondents conducting surveys. So, the best practice is to find the errors commonly used in your shop for immediate translation, and add others as needed.

To change a particular message, search for the appropriate error message using your text editor. Add your newly translated error message BEFORE the original English message, pre-pending the language letters to the message number. Your new msgfile will not compile successfully if your error message is in the wrong location in the raw file.

Example:

```
ge2602 M*** das ist alles ***
fr2602 M*** C'est tous! Au revoir! ***
sp2602 M*** Somos finito! Adios! ***
2602 M*** That is All ***
```

NOTE: Be sure to retain all Ms and %s and other programmatic formatting.

For a more systematic approach, you can compile a list of the messages you want to add or change and keep them in a separate file. You can use the supplied file “mknewmsg.spx”, installing your messages in the CfMC MSGFILE.RAW file by saying “mentor mknewmsg.spx –mknewmsg.lfl”. This run will output a new raw file that you can use to rebuild the msgfile using the MAKEMSG program. This is useful because when you get updated files from CfMC, they will not include your custom messages. If you want CfMC to ALWAYS include the messages for your language, and they are not in conflict with existing messages, you may send CfMC the message and we will incorporate it.

Create a new message file with all languages in it by saying:

```
makemsg msgfile.raw gapfile language=xx
```

This will create a file called newfile. Rename it to msgfile and place it in the cfmc/control directory.

> **NOTE:** ONLY replace the msgfile when the CfMC server is down.

Changing the language

Programmatically, you may use the !SYS,L,<language code> command:

```
{!sys,l,<lang letters>}
```

The language specified stays in effect until it is changed again or until the end of the current interviewing session.

CATI interviewers may type "L=<language code>" at any question prompt to change the language. Once this is typed, the current question redisplays in the language specified.

You have to use a different approach with webCATI interviewers: Set up a special block that sets the language and control access to it with a button on btnsbtm.tmpl.

The following is a short example:

```
{!special}
{!-AllowSuspend}
{!-Allowbackup} ``only the "next" button will show up on the special block
{!-Allowspecial}
{!-AllowTerminate}

{setlang:
\L**set language
!fld
1 \L**English
2 \L**Italian}

>rep $a=1,2;$b=en,it
{!if setlang($a)
!sys,L,$b}
>endrep
{!endspecial}
```

This would be your extra button in the btnsbtm.tmpl:

```
<TD width="10%"> <INPUT TYPE="submit" NAME="Special.x"  VALUE=" SwitchLang ">
```

Setting up the index for webSurvent

Two extra hidden inputs can be added to the index.html file: "LANGUAGE" and “QFFNAME”:

```
<input type="hidden" name"LANGUAGE" value="fr" />
<input type="hidden" name="QFFNAME" value="studyeu" />
```

Specifying the "LANGUAGE" hidden input is equivalent to having a {!sys,L,<lang code>} in your qpx: the system automatically sets the language to the value of the tag, which must be a two-digit code. However, the system knows the language right after the index page has been submitted. Therefore the program can display the appropriate language even before the respondent is in the survey (in the error.tmpl file, for example).

If you have a survey that can be undertaken in various languages and let the respondent choose the language, we recommend to have a single entry point, e.g: index.html, where the respondent chooses the language; clicking on the appropriate choice/button on index.html would lead to the chosen index page with the appropriate hidden input values for LANGUAGE and QFFNAME.

You need to add QFFNAME only if you want to keep character sets separated, for example if you are running a 77 study on 8.1+. If you run a study where the languages are all encoded to UTF-8, you do not need QFFNAME and the following considerations about QFFNAME do not apply.

You can have a different QFFNAME variable in the index file than the study code the QUOTA, PHONE, and DATA files use. This allows you to do many things such as:

Separate questionnaires for each language but the same sample, data, and quota file for a multi-language study.

Coding mode questionnaires where users are entering data and interviewers are editing or augmenting the data.

Callback studies where some users are in the first version of the questionnaire and others are in the "callback" phase. **NOTE:** The QFFNAME may have a fully qualified filename.

We recommend having as many index pages as many languages you have; the setup is fairly simple as the index pages only require the correct hidden inputs.

Let’s say you were running a multi-language study in the U.S., Europe, Japan, and the languages are English, French, German and Japanese. The indexes required would be three for the western European languages and one for the Japanese and their URL’s would look like:

-www.mysite.com/study/indexen.html

-www.mysite.com/study/indexfr.html

-www.mysite.com/study/indexde.html

-www.mysite.com/study/indexjp.html

The first three indexes would have:

```
<input type="hidden" name"LANGUAGE"    value="en" />,
<input type="hidden" name"LANGUAGE"    value="fr" />,
<input type="hidden" name"LANGUAGE"    value="de" />
```

… they also would have the same

```
<input type="hidden" name="QFFNAME" value="studyeu" />.
```

The last index, for Japanese, would have:

```
<input type="hidden" name"LANGUAGE"    value="jp" />  and
<input type="hidden" name="QFFNAME" value="studyjp" />,
```

… since you cannot have a study with both extended_ascii and shift_jis character sets.

The two studies would still point to the same fone file, quota file and write to the same tr file, using the qffname option in the header:

```
[study,32000,qffname=studyeu,…,….] for extended_ascii,
[study,32000,qffname=studyjp,….,….] for shift_jis
```

Setting up an entry point page for webSurvent

Now you can set up the entry point index.html and assume a respondent would be given username and password info to access the study (for example, myname and test, respectively) via an e-mail link. We will need to carry over this bit of information into webSurvent's index page and fill in the login details, using a simple JavaScript:

You will need a JavaScript on-click event handler on every anchor tag in index.html:

```
<A HREF=http://"www.mysite.com/study/indexen.html" onClick="append(this)">English</A>
<A HREF=http://"www.mysite.com/study/indexfr.html" onClick="append(this)">French</A>
<A HREF=http://"www.mysite.com/study/indexde.html" onClick="append(this)">German</A>
<A HREF=http://"www.mysite.com/study/indexjp.html" onClick="append(this)">Japanese</A>
```

The JavaScript is able to send the respondent to the appropriate index, carrying over the login information:

```
<script language="javascript">
 function append(thislink) {
 var loc = window.location.href;
 var locsp = loc.split("?");
 var passout = "?" + locsp[1];

 thislink.href = thislink + passout;
 }
 </script>
```

The resulting index URL for an English speaker, e.g., would then be:

```
http://www.mysite.com/study/indexen.html? name=myname&password=test
```

The respondent would eventually be routed to the appropriate index page and his/her login fields would be filled in automatically; this last action is handled by initial81.js.

Setting up the template files

WebSurvent and webCATI users need to take into consideration proper language handling in the tmpl files. Usually, only the webSurvent user will need to have a language switch in the tmpls, but what follows can apply to both webSurvent and webCATI.

If you are running a 7.7 study in 8.1+, you must have a set of tmpl files for each character set, pointing to the appropriate set of tmpl files in the index file:

```
<input type="hidden" name="STUDY_DIR"
value="/cfmc/test8.1/websurv/studies/study/extended"/>
```

The instruction above will be present in the indexen.html, indexfr.html, indexed.html.

```
<input type="hidden" name="STUDY_DIR" value="/cfmc/test8.1/websurv/studies/study/shift"/>
```

The instruction above will be present in the indexjp.html.

If you are running a UTF-8 study in 8.1 you need to have only one set of files, encoded to UTF-8

Tmpl files may contain the appropriate text for all the languages of your survey. An !html define can be used to pass the .tmpl file info about which language should be displayed. We can get the current language the survey is currently being conducted with the {!spc,7,110,2} command, like so:

{currlang:

!spc,7,110,2}

“Currlang:” will hold a two-character digit that specifies the current language. Then, you can use: {!html_define CurrentLang \:CurrentLang}

and in the .tmpl file:

```
  >if @CurrentLang = "en"

    display some English text here

  >elseif @CurrentLang = "fr"

    display some French text here

  >elseif @CurrentLang = "ct"

    display some Chinese tradtional text here

  >endif
```

You can also use defines on the .tmpl file to be more efficient. One application can be changing the buttons’ language in buttons.tmpl:

```
>if @CurrentLang = "en"

>define @next next
>define @previous  previous
>define @suspend  suspend
>define @help help

>elseif @CurrentLang = "ja"
```

```
>define @next
>define @previous    ・サー
>define @suspend
>define @help

>elseif @CurrentLang = "zh"

>define @next
>define @previous
>define @suspend
>define @help

>endif
<table border="0" align="left" cellpadding="0" cellspacing="0"
width="75%">
<tr>
<td width="20%">
<input type="submit" name="next.x" id="next.x"
style="background:#006699;color:#e3e9f0" value="   @next   ">
</td>
<td width="20%">&nbsp; </td>
<td width="20%">
>ifdef @backup_allowed
    <input type="submit" name="previous.x" id="previous.x"
style="background:#006699;color:#e3e9f0"   value=" @previous ">
>endif
</td>
<td width="20%">
>ifdef @suspend_allowed
    <input type="submit" name="suspend.x" id="suspend.x"
style="background:#006699;color:#e3e9f0"  value=" @suspend ">
>endif
</td>
<td width="20%">
<input type='button' name='help' id='help'
style='background:#006699;color:#e3e9f0' value='   @help   '
onclick='pop_help();'>
</td></tr></table>
```

You will also need to specify the appropriate character encoding in the META tag this way in either header.tmpl or pagetop.tmpl:

```
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
```

Even if we explicitly specify the encoding in header.tmpl or pagetop.tmpl, the Apache Web server can still override its value. The file httpd.conf has an entry called AddDefaultCharset. Its values can be: On|Off|charset. This should override any charset specified in the body of the response via a META element, though the exact behavior is often dependent on the user's client configuration.

A setting of AddDefaultCharset Off disables this functionality. AddDefaultCharset On enables a default charset of iso-8859-1.

In general, if we run multi-language studies, we want to have the AddDefaultCharset parameter set to off.

You can find more information on this here:

http://httpd.apache.org/docs/2.0/mod/core.html#adddefaultcharset

You might have a multi-language qpx with languages that, even if within a character set, might span across different encodings. For example, a qpx that contains, for example, English and Hungarian, will have the characterset set to extended_ASCII. English falls under the iso-8859-1 encoding, whereas Hungarian falls under iso-8859-2.

Then, you need to switch between these two encodings in either the header.tmpl or pagetop.tmpl.

Your qpx might look like:

```
{!HTML_DEFINE METATAG }

{LANG:
What language?
!FLD,
1 English
2 Hungarian }

{
!IF LANG(2)
!GOTO HUNG}

{ENGL: !GOTO }
{!HTML_DEFINE METATAG <meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1">}
{!GOTO START }
```

```
{HUNG: !GOTO }
{!HTML_DEFINE METATAG <meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-2">}
{!GOTO START }

{START:
!Goto}
```

The tmpl will look like:

```
>ifdef@metatag
@METATAG
>endif
```

Then, you will have switches that control which language to display on the page:

```
>ifdef @fr
display some French text here
>endif

>ifdef @de
display some German text here
>endif
```

Once again, if your languages are UTF-8 encoded, you will not need to switch meta tag as you will only need :

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
```

The error message in error.tmpl (@error_text) is already translated, but in order to have the whole page in the current language you would need a a custom error.tmpl for every character set (if you are in 7.7 or running a 7.7 prepped study in 8.1+). In order to do this you can setup a different STUDY_DIR for every character set, which will contain the appropriate error.tmpl.

Below an example that shows how you could set error.tmpl for the extended_ascii character set, with two languages. The example also takes into consideration the possibility of an index that might not have the “language” hidden input tag set.

```
>if "@language"="en"
<h2> An Error Has Occurred! </h2>
<font color="#aa00ff"> @error_text </font>
<br /><br />
<p />
If this problem persists, please contact us by clicking on this e-mail link:
<p />
<a href="mailto:@mailto?subject=Study:@studycode,User:@name,Password:@password,
ErrorCode @error_code">@mailto</a>
<p />
Please describe the problem in as much detail as possible.
>endif
```

```
>if "@language"="it"
<h2> Si è prodotto un errore! </h2>
<font color="#aa00ff"> @error_text </font>
<br /><br />
<p>
Se il problema persiste, contattateci cliccando sul link email:
</p><p>
<a href="mailto:@mailto?subject=Study:@studycode,User:@name,Password:@password,
ErrorCode @error_code">@mailto</a>
</p><p>
Per cortesia descrivete il problema nel maggiore dettaglio possibile.
</p>
>endif
```

Once again, if you are working in UTF-8 you can have only one UTF-8 encoded error.tmpl in the study directory.

Setting up JavaScript error messages

Create a copy of user_settings.js for each language. Keep in mind that special character encoding must be done in Unicode (UTF8) in order to display properly.

Set an !html_define in your spec for the language being used:

```
{lang: hide

!fld

1 English

2 Spanish

3 French }

{!if lang(<>1)

!goto nextlan1}


{!html_define English}
```

```
{nextlan1
!if lang(<>2)
!goto nextlan2}

{!html_define Spanish}

{nextlan2
!if lang(<>3)
!goto donelang}

{!html_define French}

{donelang:
!goto }
```

When you call user_settings.js in your header.tmpl or pagetop.tmpl, use >ifdef to call in the correct version of the file.

```
>ifdef English
<script src="/cfmcweb/js/user_settings_english.js" type="text/javascript"></script>
>endif

>ifdef Spanish
<script src="/cfmcweb/js/user_settings_spanish.js" type="text/javascript"></script>
>endif

>ifdef French
<script src="/cfmcweb/js/user_settings_french.js" type="text/javascript"></script>
>endif
```

And, then, call cfmc_ws81.js as you normally would.

Another way to achieve the same functionality with less code is:

```
 {CurrentLang:
 !spc,7,110,2}

{!html_define CurrentLang \:CurrentLang}
```

and then in pagetop.tmpl or header.tmpl:

```
>if CurrentLang = "en"

<script src="/cfmcweb/js/user_settings_english.js"
type="text/javascript"></script>

>elseif CurrentLang = "sp"

<script src="/cfmcweb/js/user_settings_spanish.js"
type="text/javascript"></script>

>elseif CurrentLang = "fr"

<script src="/cfmcweb/js/user_settings_french.js"
type="text/javascript"></script>
  >endif
```

Building the multi-language fone file

Building a multi-lingual phone file is no different than building a non multi-lingual one, but it is useful to make some considerations.

First off, in versions 7.7 and 8.1 you have 4900 *columns* available in the fone file text area. It is important to note here that there is not a one-to-one correspondence between a column and a character. This is true only if you have sample with extended ASCII characters. In general, a column in the text area means a byte. If you run a job which has sample text in shift_jis, multi-byte or UTF-8 with Asian languages you will find that the sample fields use more space than what a visual inspection might suggest.

A step-by-step guide to building a multi-language sample file might be the following:

Save from Excel following the steps outlined before

Once you have a text file from Excel, for example a tab delimited one, you can run a spec like this on it:

```
~input from_excel.txt delimiter=tab delimit_map ascii length=1000

~output fixed_format_file.txt ascii writenow

~end
```

The spec above will ouput a map of the fields (called <specname>.out), as well as a fixed width output file (fixed_format_file.txt).

If you have non ASCII characters, a quick visual inspection of the output file and the map tells us that there is a discrepancy between the max number of columns the map shows and the mex number of "characters" (in this case ideograms probably). This tells us the map is referring to the number of bytes and not ideograms.

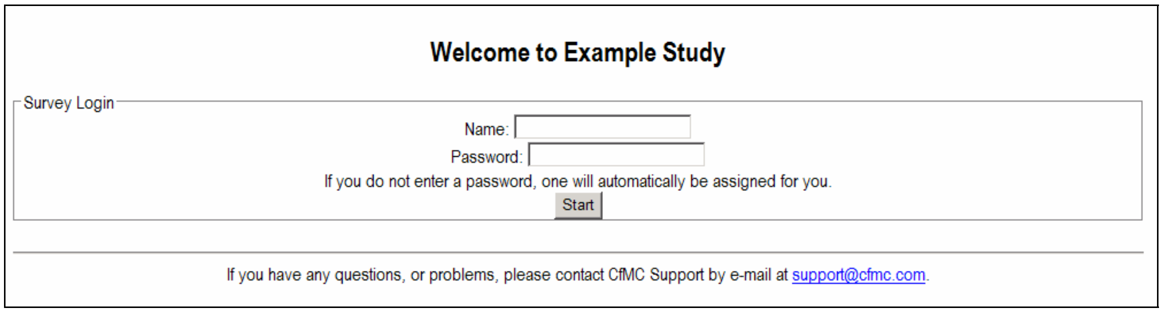
You can then run WSMOVER to create a raw sample file that FONEBULD can read, using the information from the map.

Appendix D

WEBSURVENT QUESTIONNAIRE SOURCE CODE

WebSurvent generates the source code for all screens from the variables you specified in the WEBUTIL process (*Chapter 7*) and from webSurvent template files (*Chapter 8*).

This example is provided to help Web designers understand how webSurvent works:



Welcome to Example Study

Survey Login

Name:

Password:

If you do not enter a password, one will automatically be assigned for you.

Start

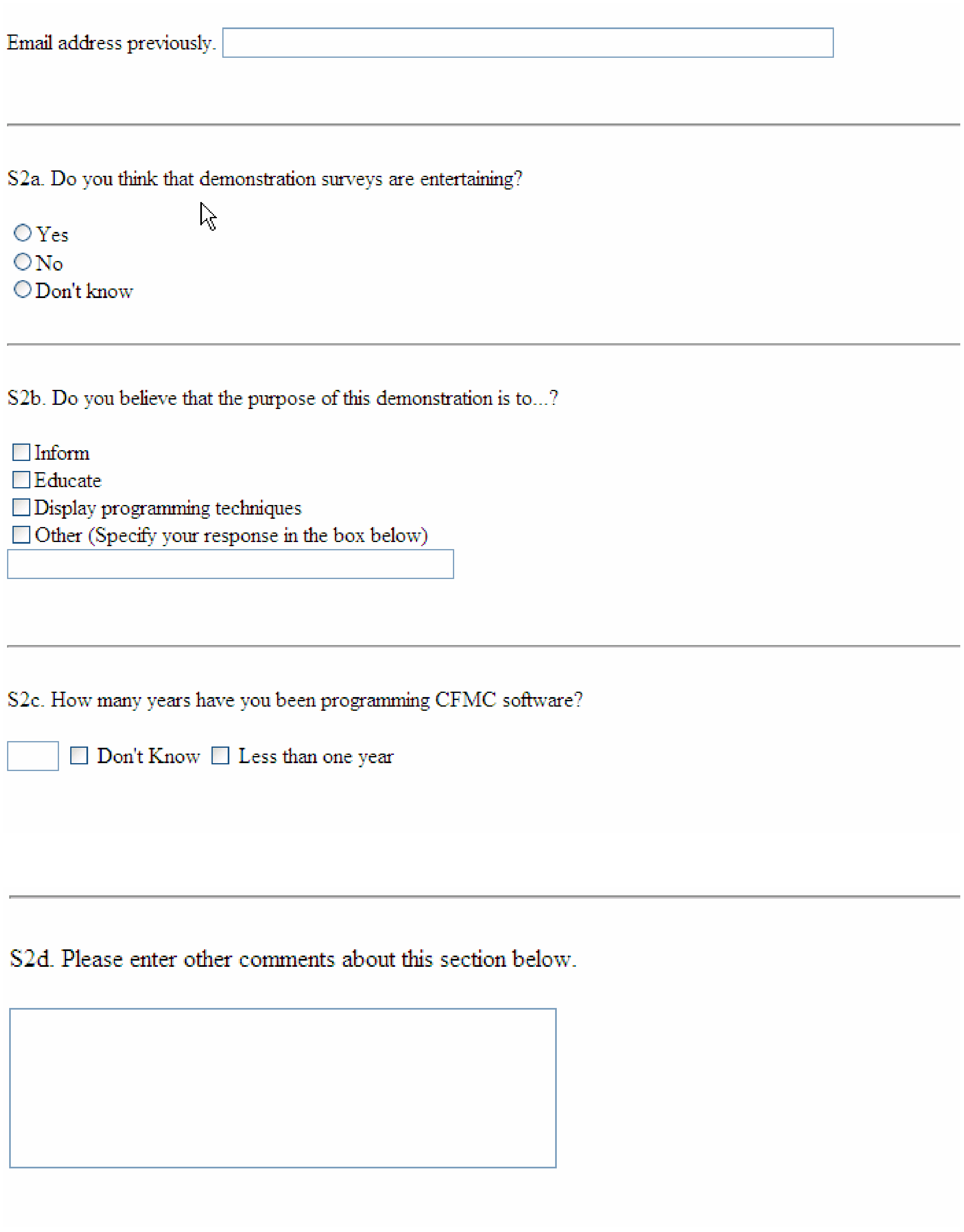
If you have any questions, or problems, please contact CfMC Support by e-mail at support@cfmc.com.

```
<html>
<head>
<title>Example Study</title>
</head>
<body bgcolor="white" text="black" />
<center />
<h2>Welcome to Example Study<br />
</h2>

<form name="cfmclogin" method="post" action="/cgi-bin/cfmccgi/websrv81.cgi">

<!-- System Specific (Probably Won't Change) -->
<input type="hidden" name="CFMC" value="/cfmc/test8.1/"/>
<input type="hidden" name="CFMCCFG" value="/cfmc/test8.1/ipcfiles/"/>
<!-- Tmpl locations -->
<input type="hidden" name="STUDY_DIR" value="/cfmc/test8.1/websurv/studies/demo/"/>
<input type="hidden" name="TMPLS_DIR" value="/cfmc/test8.1/websurv/tmpl/WS_default/"/>
```

```
<!-- Other Study Specific items -->
<input type="hidden" name="STUDYCODE" value="demo"/>
<input type="hidden" name="USE_PASSWDS" value="NO"/>
<input type="hidden" name="MAILTO" value="support@cfmc.com"/>

  <div>
    <fieldset>
    <legend>Survey Login</legend>
    <label for="name">Name:</label>
    <!-- Use this instead of "Name:" line if everyone's named "Respondent":
         <input type="hidden" name="name" value="Respondent">
         -->
    <input type="text" name="name" id="name" tabindex="1" value="" />
    <br />
    <label for="password">Password:</label>
    <input type="text" name="password" id="password" tabindex="2" />
    <br />
    If you do not enter a password, one will automatically be assigned for you.
    <br />
  <span><input class="submit" type="submit" name="start" value="Start" tabindex="3" /></span>
  </fieldset>
  </div>

</table>
</form>

<p /><hr />If you have any questions, or problems, please contact CfMC Support by e-mail at
<a href="mailto:support@cfmc.com?subject=demo indexnp">support@cfmc.com</a>.

<script src="/cfmcweb/js/user_settings.js" type="text/javascript"></script>
<script src="/cfmcweb/js/initial81.js" type="text/javascript"></script>

</body>
</html>
```

Here is an example of the screen and source code from inside a survey.

The following section shows how a questionnaire would look with simple HTML and the use of a grid to display multiple questions on screen.



Email address previously.

S2a. Do you think that demonstration surveys are entertaining?

- ○ Yes
- ○ No
- ○ Don't know

S2b. Do you believe that the purpose of this demonstration is to...?

- ☐ Inform
- ☐ Educate
- ☐ Display programming techniques
- ☐ Other (Specify your response in the box below)

S2c. How many years have you been programming CFMC software?

☐ Don't Know ☐ Less than one year

S2d. Please enter other comments about this section below.

The source code for the example above follows:

```
<HEAD>

<!-- begin CFMC Template File: /cfmc/rel8.1/websurv/studies/exam81/header.tmpl -->

<script src="/cfmcweb/js/user_settings.js" type="text/javascript"></script>
<script src="/cfmcweb/js/cfmc_ws81.js" type="text/javascript"></script>

<title>Websurvent Survey </title>



<!-- end CFMC Template File: /cfmc/rel8.1/websurv/studies/exam81/header.tmpl -->

</HEAD>

<!-- begin CFMC Template File: /cfmc/rel8.1/websurv/studies/exam81/bannrtop.tmpl -->

<BODY  bgcolor="white" text="black" >

<!-- end CFMC Template File: /cfmc/rel8.1/websurv/studies/exam81/bannrtop.tmpl -->

<FORM name="CFMC" id="CFMC" METHOD="POST" ACTION="/cgi-
bin/cfmccgi/websrv81.cgi">
<!-- begin CFMC hidden fields -->
<INPUT TYPE="HIDDEN" NAME="NAME" VALUE="test1234">
<INPUT TYPE="HIDDEN" NAME="PASSWORD" VALUE="test1234">
<INPUT TYPE="HIDDEN" NAME="CFMC" VALUE="/cfmc/rel8.1/">
<INPUT TYPE="HIDDEN" NAME="CFMCCFG" VALUE="/cfmc/rel8.1/ipcfiles/">
<INPUT TYPE="HIDDEN" NAME="STUDYCODE" VALUE="exam81">
<INPUT TYPE="HIDDEN" NAME="STUDY_DIR"
VALUE="/cfmc/rel8.1/websurv/studies/exam81/">
<INPUT TYPE="HIDDEN" NAME="TMPLS_DIR" VALUE="/cfmc/rel8.1/websurv/tmpl/">
<INPUT TYPE="HIDDEN" NAME="USE_PASSWDS" VALUE="OPEN">
<INPUT TYPE="HIDDEN" NAME="CGI_PHASE" VALUE="0">
<INPUT TYPE="HIDDEN" NAME="FONERECNUM" VALUE="868">
```

```
<INPUT TYPE="HIDDEN" NAME="survent_pid" VALUE="21181">
<INPUT TYPE="HIDDEN" NAME="survent_ldev" VALUE="10016">
<INPUT TYPE="HIDDEN" NAME="survent_port" VALUE="25016">
<INPUT TYPE="HIDDEN" NAME="server_ldev" VALUE="901">
<INPUT TYPE="HIDDEN" NAME="HTML_SEQ" VALUE="2">
<INPUT TYPE="HIDDEN" NAME="CFMCVAL" VALUE="20081015 9001 0001 0127 0500
0500 0500 0100 0500 0500 0000 M0050DA6FFF72">
<INPUT TYPE="HIDDEN" NAME="QFFNAME" VALUE="/cfmc/rel8.1/qff/exam81.qff">
<!-- end CFMC hidden fields -->


<!--CFMCSTARTGRID :630,6.30: -->
<b>The following section shows how a questionnaire would look with simple html and the use of a
grid to display multiple questions on screen.
</b><br><br>
</label>
<br><br>
<!--CFMCSTARTPAGE :633: -->
<br><hr><br>
<label for="EMAIL2">
Email address previously.
</label>
<INPUT TYPE="text" NAME="var_EMAIL2" id="EMAIL2" SIZE="70" maxlength="70"
VALUE="">
<br><br>
<!--CFMCSTARTPAGE :636: -->
<br><hr><br>
S2a. Do you think that demonstration surveys are entertaining?
<br><br>
<INPUT TYPE="radio" NAME="catr_S2A" id="S2A_1" VALUE="1"><label
for="S2A_1">Yes</label><br>
<INPUT TYPE="radio" NAME="catr_S2A" id="S2A_2" VALUE="2"><label
for="S2A_2">No</label><br>
<INPUT TYPE="radio" NAME="catr_S2A" id="S2A_3" VALUE="3"><label
for="S2A_3">Don't know</label><br>
<!--CFMCSTARTPAGE :639: -->
<br><hr><br>
```

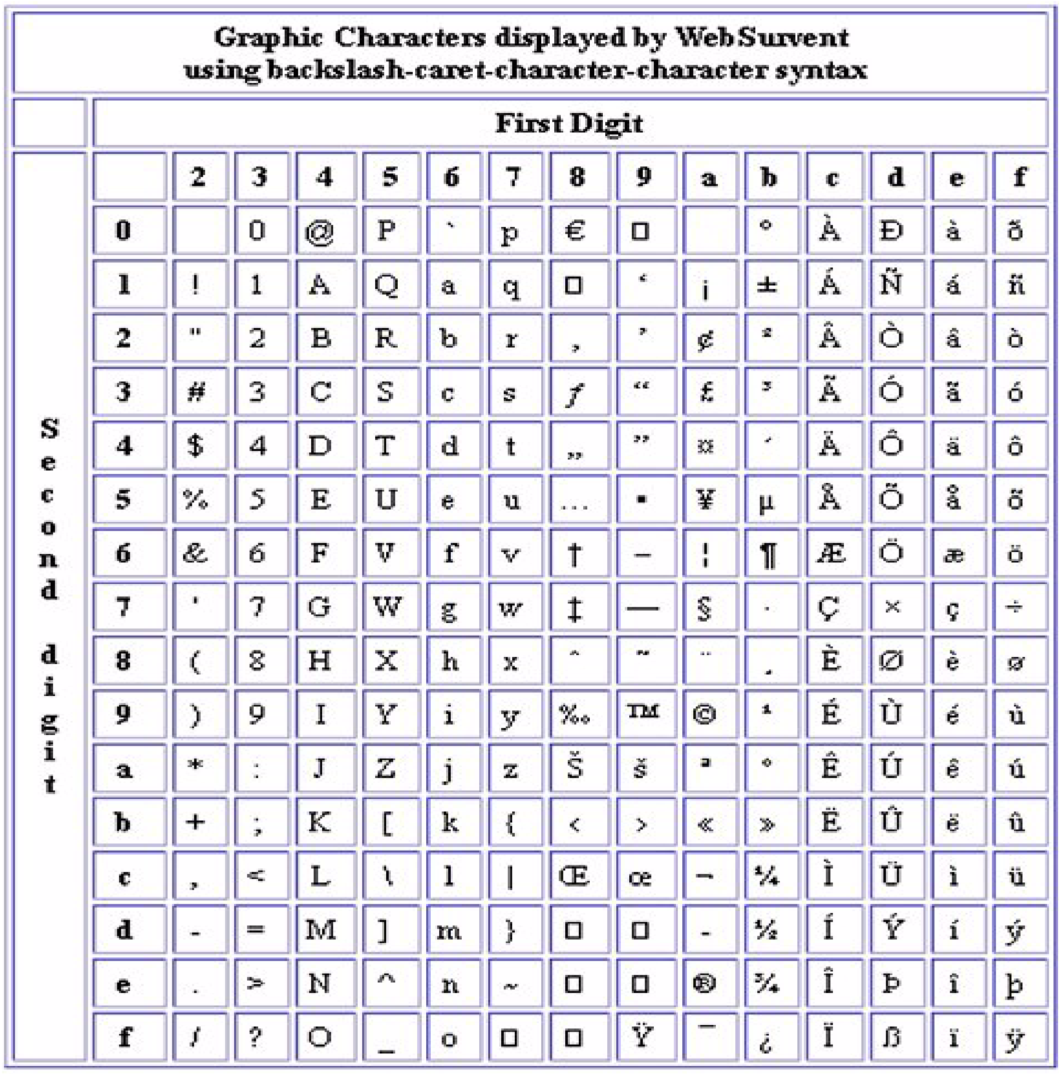
```
S2b. Do you believe that the purpose of this demonstration is to...?
<br><br>
<INPUT TYPE="checkbox" NAME="catm_S2B" id="S2B_1" VALUE="1"><label
for="S2B_1">Inform</label><br>
<INPUT TYPE="checkbox" NAME="catm_S2B" id="S2B_2" VALUE="2"><label
for="S2B_2">Educate</label><br>
<INPUT TYPE="checkbox" NAME="catm_S2B" id="S2B_3" VALUE="3"><label
for="S2B_3">Display programming techniques</label><br>
<INPUT TYPE="checkbox" NAME="catm_S2B" id="S2B_4" VALUE="4"><label
for="S2B_4">Other (Specify your response in the box below)</label><br>
<!--CFMCSTARTPAGE :643: -->
<label for="S2BOTH">
</label>
<INPUT TYPE="text" NAME="var_S2BOTH" id="S2BOTH" SIZE="50" maxlength="50"
VALUE="">
<br><br>
<!--CFMCSTARTPAGE :647: -->
<br><hr><br>
<label for="S2C">
S2c. How many years have you been programming CFMC software?
<br><br>
</label>
<INPUT TYPE="text" NAME="num_S2C" id="S2C" VALUE="" SIZE="2" maxlength="2">
<INPUT TYPE="checkbox" NAME="rnum_S2C" id="S2C_DK" VALUE="DK">
<label for="S2C_DK">Don't Know</label>
<INPUT TYPE="checkbox" NAME="rnum_S2C" id="S2C_LT" VALUE="LT">
<label for="S2C_LT">Less than one year</label>
<br><br>
<!--CFMCSTARTPAGE :650: -->
<br><hr><br>
<label for="S2D">
S2d. Please enter other comments about this section below.
<br><br>
</label>
<TEXTAREA NAME="tex_S2D" id="S2D" ROWS="5" COLS="40">
</TEXTAREA>
```

```
<br><br>
<!--CFMCSTARTPAGE :652: -->
<!--CFMCENDGRID :652: -->



<!-- begin CFMC Template File: /cfmc/rel8.1/websurv/studies/exam81/btnsbtm.tmpl -->


<br clear="all" />
<hr />


<table border="0" align="left" cellpadding="0" cellspacing="0" width="75%">
<tr>
<td width="20%">
<input type="submit" name="next.x"  style="background:#006699;color:#e3e9f0" value="next">
</td>
<td width="20%">&nbsp; </td>
<td width="20%">
<input type="submit" name="previous.x" style="background:#006699;color:#e3e9f0"
value="previous">
</td>
<td width="20%">
<input type="submit" name="suspend.x" style="background:#006699;color:#e3e9f0"
value="suspend ">
</td>
<td width="20%">
<input type="button" name="help" style="background:#006699;color:#e3e9f0" value="help"
onclick="pop_help();">
</td></tr></table>



<!-- end CFMC Template File: /cfmc/rel8.1/websurv/studies/exam81/btnsbtm.tmpl -->
```

```
</FORM>

<!-- begin CFMC Template File: /cfmc/rel8.1/websurv/studies/exam81/bannrbtm.tmpl -->

<br />
<div>
 <div>
  <p>Questions? Email <a
href="mailto:support@cfmc.com?subject=question,study:exam81,username:test1234,password:test
1234">support@cfmc.com</a></p>
  <div id="statbar" name="statbar"></div>
 </div>
</div>

<form name="statbar">
<input type="hidden" name="statbar_perc" value="-1">
</form>



<!-- end CFMC Template File: /cfmc/rel8.1/websurv/studies/exam81/bannrbtm.tmpl -->
```

Appendix E

GRAPHIC CHARACTERS



Graphic Characters displayed by WebSurvent using backslash-caret-character-character syntax

	First Digit													
Second digit	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0		0	@	P	`	p	€	□		°	À	Ð	à	ð
1	!	1	A	Q	a	q	□	‘	¡	±	Á	Ñ	á	ñ
2	"	2	B	R	b	r	‚	’	¢	²	Â	Ò	â	ò
3	#	3	C	S	c	s	ƒ	“	£	³	Ã	Ó	ã	ó
4	$	4	D	T	d	t	„	”	¤	´	Ä	Ô	ä	ô
5	%	5	E	U	e	u	…	•	¥	µ	Å	Õ	å	õ
6	&	6	F	V	f	v	†	–	¦	¶	Æ	Ö	æ	ö
7	'	7	G	W	g	w	‡	—	§	·	Ç	×	ç	÷
8	(	8	H	X	h	x	ˆ	˜	¨	¸	È	Ø	è	ø
9	)	9	I	Y	i	y	‰	™	©	¹	É	Ù	é	ù
a	*	:	J	Z	j	z	Š	š	ª	º	Ê	Ú	ê	ú
b	+	;	K	[	k	{	‹	›	«	»	Ë	Û	ë	û
c	,	<	L	\	l	|	Œ	œ	¬	¼	Ì	Ü	ì	ü
d	-	=	M	]	m	}	□	□	-	½	Í	Ý	í	ý
e	.	>	N	^	n	~	□	□	®	¾	Î	Þ	î	þ
f	/	?	O	_	o	□	□	Ÿ	¯	¿	Ï	ß	ï	ÿ

This chart shows the graphic characters associated with webSurvent. Hexadecimal values may be used to display a variety of graphic characters in survey text.

Enter \^ and the two-digit hexadecimal value corresponding to the desired symbol. Some of these characters may be needed for multilingual Web surveys. The hexadecimal characters will display as shown in DOS. On other computers, it depends on which characters the terminal and operating system support for that terminal type, and which character set the terminal is emulating. Refer to the example questionnaire CHARS^QPX in the Survent subdirectory of CfMC (UNIX) or the Survent group of account CfMC (MPE) to see what these escape sequences look like on your system.

NOTE: Columns 0 and 1 (first digits) are not pictured because they are not applicable in webSurvent.

CfMC recommends that you use your browser to search for information regarding Graphic Characters for your particular Operating System.

Appendix F

ADDING A STATUS BAR

A status bar may be added to your webSurvent screen to show the respondent how far along they are in the survey. By default, Survent will provide a percent based on the question number you are on, but you can specifically control the percent that displays.

The following syntax is used to update the value on the status bar:

Syntax:

{!STATUS_BAR <##/qq/val=\|statusbr|>}

Where:

- set the statusbar value to the specific number (##). This value MUST be between 1-100

val=\|<exp q> - assign a value calculated in the questionnaire from an expression (!exp). This value MUST be between 1-100.

qq - put the actual percent you have moved through the questionnaire (current question number/total questions)

The program will update the percentage each time it sees a new STATUS_BAR "##", "qq", or "val=" option. You would use the "val=\|statusbr|" to specifically set a value, if, for instance, you were in the middle of a rotation block and needed to calculate how far along you had gotten, you would add the following:

{statusbr: .3 set an expression to zero

!expr,,0 } Put this in front of each grid or question in the rotate

{[statusbr] increase the statusbar expression by percentage you want

!expr,,statusbr + 5 } for a grid or question

{!statusbar val=\|statusbr| } pass the the value to the JavaScript

The controls for the status bar are located in your user_settings.js. A few options are available:

Enabling the status bar and ticker:

```
var user_use_statusbar = true; //true or false
var user_use_statusticker = true; //true or false
```

The default images/colors of the status bar are light orange and dark orange. The status bar uses one-pixel images – you can choose from several files in the /Pixels directory, or create your own to suit your color scheme. These can be customized by changing the following paths:

```
var user_statusbar_done_image = "/cfmcweb/images/Pixels/ff9966.gif";
var user_statusbar_left_image = "/cfmcweb/images/Pixels/cc6666.gif";
```

To control the width, height, text and alignment of the statbar:

```
var user_statusbar_text = "<FONT SIZE='-1' color='black'><i> Progress </i></FONT>";
var user_statusbar_width = 150;
var user_statusbar_align = "right";
var user_statusbar_thickness = 10;
```

The variables listed above are the default settings, and will be applied if you do not change them or do not call user_settings.js in your header. You can also use style sheet elements within your .tmpl files, to further customize the status bar table. See *Appendix H* for more information on customizing your template files.

Appendix G

THE WEBSURVENT WEB SHELL

Web Shell, version 2.0, is an automated process of creating specifications for Web surveys and WebSurvent in particular. The shell contains presets that are integral to the operation of WebSurvent. You can modify these settings as you see fit, according to your types of Web interviews and your company specifications.

Tracking information

Make sure you fill out the tracking information at the top, because this can be very helpful in managing projects and tracking problems. Version, Qdate and wave will be stored in data as EXPR questions with the same names.

Structural requirements

These meta commands set up some base structure:

>Purgesame - sets system to overwrite any existing files created in the compile. By default, a backup version of the file will be created instead.

>LocFormat 1 – defines how data locations will be displayed. In this case they will be displayed as absolute columns. Change this to 1/1 if you prefer a card-column format. This command does not change how data is stored, only how data locations are displayed.

>FillDefinesInQuotes – Replaces define calls with the appropriate text in auxiliary files.

Setup of defines

Numerous defines have been set up at the top of the shell. They can be used to create compile switches to control the flow of the survey, to insert standardized text and to insert html code. In the latter case, these defines are set up so that they do not get passed into the Def or db files and therefore do not show up in any Mentor runs.

Compile switches

All compile switches are “On” if they are not commented out and “Off” if they are.

The logic behind this is handled with the command:

```
>ifdef @definename
"Statement"
>endif
```

```
or
>ifdef @definename
"Statement"
>else
"Statement"
>endif
```

If the define is *not* commented out, the >ifdef condition evaluates as true. If it *is* commented out, it evaluates as false.

>define @Saveterm – When turned on, all terminates will be saved to data, either upon a suspend or a completion.

>define @Test - When turned on, any password that begins with "test" is treated as a live survey. When turned off, passwords beginning with "test" will not be saved to data, either as a suspend or a complete and will not update any quotas. This switch can also be used to allow for certain questions to be asked only while testing. For example, if you had a sample-driven questionnaire, you could display a question while testing and pull the information from the fone file when live.

Example:

```
>ifdef @Test
{samptype:
Please enter the sample group you would like to test.
!fld
1Sample Group 1
2Sample Group 2
3Sample Group 3 }
>else
{samptype:
!phone,g,151,1}
>endif
```

>define @debug - Can be used to display debug information while you are testing. The advantage of using this in conjunction with the @test define is that you can turn the debug on while you are testing and turn it off while your client tests. For example, if you have a series of hidden questions that you would like to see while you are testing, but your client doesn't need to see, you can do the following.

```
>ifdef @debug
{
```

```
The value of hidden question 1 is \|q1hide|
The value of hidden question 2 is \|q2hide|
!disp }
>endif
```

>define @statout - Can be used to write the percentage of the survey completed to the fone file when respondents suspend. In order to do this you must use {!statusbar val=\|statbar|} to track/store the status bar percentage. IT WILL NOT WORK IF YOU ARE USING {!statusbar qq}. You also need to set the fone file column(s) you want to store this information in by setting the value of >Define @Statloc.

>define @Quotaout – As above, this can be used to store the quota group assigned into the fone file. You must assign a value to the EXPR question quotagrp for each group you want stored.

For example:

```
{!if q1(1)
!quota,,,q1_1}
{[quotagrp]
!if q1(1)
!expr,z,,001 }
{!if q1(2)
!quota,,,q1_2}
{[quotagrp]
!if q1(2)
!expr,z,,002 }
```

You also need to define the column in your fone file to output this by setting the value of >define @quotaloc.

Text defines

Text defines can be used to insert the desired text anywhere in your questionnaire.

For example, anywhere you type @disp in your spec, the text “Click on “Next” to continue” will be inserted. Structurally, these are surrounded with back slash commands for additional control or formatting. They all use \+* so that they will not be output to files used by Mentor. They all also are set to appear in italics \I and the color red \CR. These can be changed as desired.

Html defines

Just like the text defines, these can be used to insert commonly used html tags. The advantage to using these is that, like the text defines, they are formatted so as to not output to files used by Mentor. Most of these can be used by simply referencing the define (@br or @table); the only ones that require additional information are explained below:

```
>define @tdlw \+<td align=left width=
>define @tdrw \+<td align=right width=
>define @tdcw \+<td align=center width=
>define @thlw \+<th align=left width=
>define @thrw \+<th align=right width=
>define @thcw \+<th align=center width=
```

These give you a way to set a column width using some simple shorthand. When using these defines, a width definition should be included. This is done with the syntax:

```
@tdlw~"60%">\*
```

In this case the actual tag created would look like:

```
\+<td align=left width="60%">\*
```

The header statement

The header statement in this shell primarily sets some basic settings higher than their standard defaults (specwid, screenlines, MaxQuestionsize, etc …).

CfMC has found that Web surveys consistently require larger defaults in these areas. The defaults created by this header are:

Data case length	= 25,000 columns
Text area	= 10,000 columns
Spec width	= 300
Screenlines	= 500
MaxQuestionSize	=64000 (the maximum allowed)
MaxQFileSize	=5 (allows a maximum qff file size of 5 megabytes)
CaseID	=1.6 (place case id in column 1 with a default size of 6 characters)
AnswerLength	=60000 (Sets the size of the answer array to its maximum of 60,000 bytes.)

FoneTextLength	=900 (Sets the user area of the fone file to its maximum size of 900).

These can be changed as needed.

Additional setup

{!hardcopy ShowSameas} - creates a studyname.hrd file with sameas response categories written out.

{!hardcode} – hardcodes the data locations in the QSP file.

{!HTMLCheckboxes=500}/{!HTMLRadioButtons=500} – is the default for radio buttons/check boxes vs. drop-down boxes. Any fld/cat question with more than 500 responses will be presented in a drop-down box. Any fld/cat question with fewer than 500 responses will get check boxes or radio buttons. This just acts as a default and can be changed anywhere within your survey.

{!-AllowTerminate} - Turns the terminate function off, which means no terminate button will be presented. We've found that using a terminate button on the Web can create more problems than it solves. If you want to use a terminate button, then just comment out this line.

Statcode - Hardcoded into column 10 for a length of 3, this can be used to store a final status into the data itself. This is particularly useful if you are saving terminates to data as well. The usage is quite simple. Anytime you assign a final status (complete, terminate, over-quota, etc …) gen the appropriate code into this question. Generally, we recommend using the same codes you assign in your fone file. For example, on completion the command {!,gen,a,statcode,001} puts a 001 into this location.

Questionnaire info and markers

DateTim – stores the date and time the survey was started. Qinfo switches store information about the spec itself. Of particular importance is the Prep-Time variable as it stores the date and time the spec was complied. If there is a confusion as to which version of a survey was used this can be very helpful.

Rotloc – Rotloc can be used to store the rotation seed for various rotation subtypes. The advantage of using this is if you want to set a rotation order, you can write the seed out to one of these and then on subsequent rotations reference that seed.

```
{!rotate,s,,rotloc}
Series of questions …
{!endrotate}
```

Any future rotates you want in the same order can be referenced in the same way.

Check – Compscrn - Can be used as binary switches. If a particular event occurs, gen a 1 into these questions. Then, on preceding logic, you can use {!if check(1)} or {!check<.>1}. Testflag is already used in the shell. If a password begins with "test" then this flag will be set to 1.

Fone file commands

As defined in the shell, password, username and e-mail address will be pulled into data from the fone file. Any other phone,g statements can be inserted here as well. Also defined in the shell: any record will be initially assigned a phone status of 121 (Unspecified callback). Real fone file statuses should be set as needed. The idea here is that if you see a record with status 121 in the fone file no other status ever got assigned and something is wrong with that record.

Setting up the chkfone expr

In the course of doing Web surveys, CfMC has found that respondents we wish to terminate will often close their browsers before the process has been completed. This results in suspended interviews and all of the data collected for them is saved into a resume file. In order to avoid this problem, you can use chkfone to set a fone status. Then, logic already existing in the suspend block will handle them as terminates rather than suspends. The example of how to use this follows:

```
{S1:
Some terminate question.
!fld
1 Yes (continue)
2 No (Terminate) }
{!if s1(1)
!goto S2}
"set statcode to a terminate status
{!if S1(2)
!gen,a,statcode,020 }
"set phone status to the same terminate status
{!if S1(2)!phone,s,20 }
"update your quota
{!if S1(2)
!quota,,q1_term,1,now}
"set a data variable to match the fone status.
{!if S1(2)
[chkfone]
!EXPR,Z,chkfone+FoneStatus() }
{
!IF S1(2)
Unfortunately, you did not qualify for this survey. Thank you very much for your time.
Those are all the questions I have for you.
!display}
```

If they click on next question it will behave as normal. If they just closed their browser they will eventually suspend.

```
{!if S1(2)
!goto hangup}
```

Genscrn and Gencomp

These are used to find out:

a) that a respondent passed the screener

b) that the survey has been completed.

The Terminate block

Again, we do not recommend allowing terminate in Web surveys. However, the terminate block can be used as a single location to process screener terminates.

The key elements are:

```
>IfDef @Test
>Else
{!If TestFlag(1)
!SPC,B }
>EndIf
```

If a survey is live (the @test define is commented out), and a test password is being used, then the record will terminate at this point with no data being saved.

The following assigns different codes based on whether the respondent has passed the screener or not.

```
{!If [CompScrn^^NB] screener terminate
!Phone,S,004 }
{!If [CompScrn^^B] post-screener terminate
!Phone,S,015 }
{!If [CompScrn^^NB]
!Gen,A,StatCode,004 }
{!If [CompScrn^^B]
!Gen,A,StatCode,015 }
```

```
{!If [CompScrn^^NB]
!Quota,,Term_in_Prog,1,Now }
{!If [CompScrn^^B]
!Quota,,Term_Screen,1,Now }
{!Goto Hangup }
```

The Suspend block

The suspend block starts with the logic based on chkfone. If chkfone is greater than 1 and less than 100, then a case ID is assigned, the record is written to data, the fone status - if it is missing - will be assigned to the value of chkfone and the survey is terminated.

After this, the qq number or label of the last question asked is stored. A flag (susphide) is set to 1, so you can always tell if the suspend block has been accessed.

Again, the ifdef test logic is checked and if it is live using a test password, no resume file is written and the survey is ended.

The defines @statout and @quotaout are handled. If these defines are turned “on,” then this information is written to the phone file. Finally, a check is done to see if the record is auto-suspending (the browser was closed or connection lost) or if a normal suspend process has been followed. For auto-suspends, a phone status of 108 is set. If there is a “normal” suspend, then the phone status is set to 105.

WebSurvent now places suspended interviews in the "Never Schedule" stack 338 and gives them a status code of 156. This stack has the feature that numbers are available to be called but will be held there until you specifically ask for them. This is true of suspended or auto-suspended interviews. You can still override the status for suspended interviews if you'd like by using !PHONE,S,### in the {!SUSPEND} block to set the status.

The Resume block

Again, a marker is set if the resume block has ever been accessed. The resume date/time and the time the current qff file was compiled are stored. The quota for resumes is incremented if live and the password does not begin with “test”.

The end

The following things are resolved upon the completion of a survey

The routing for Saveterm is set. If you always use a {!goto hangup} after any terminate points, then they will be routed here and handled appropriately based on whether @saveterm is turned on or off. Again, a test for live surveys using test passwords is processed, and all test passwords will be terminated at this point.

Timing questions are set: LofIS stores the overall time in the survey in seconds. LofIM stores the same converted to minutes.

A completed quota is updated and the quotagrp written to the phone file (if quotaout is turned on).

The number of overall and consecutive backups is gathered and stored. The variable totback stores the total number if times a respondent backed up. The variable consback stores the maximum number of screens respondents backed up through at any one time.

Finally, the phone status is set to a complete status of 1.

Appendix H

JAVASCRIPTS AND INITIAL81.JS

JavaScripts can be used in conjunction with webSurvent and webCATI in several different ways. But the most basic usage of these Scripts is to make the individual pages more dynamic. Its use can be as simple as popping up text or an image if a respondent passes over a certain element of the page or as complex as checking responses and/or creating dynamic skip patterns as respondents answer questions on the page. JavaScript can be used to send respondents to a specific URL upon completion of the survey, to create pop-up help windows, to disable a respondent's ability to right click in a survey.

There are several JavaScripts that are included with webSurvent, although none are required by the software. You can run webSurvent without any JavaScript whatsoever if you choose. All scripts can be found in the Web area JavaScript directory or in /cfmcweb/js/.

You can get regular updates to these scripts and their documentation by sending an e-mail to getfiles@cfmc.com with the subject "show files". You can check which version of the scripts you have by looking at the comments at the top of each script. If you run into problems using these scripts or have a need for a script that is not included you can send email to support@cfmc.com.

Finally, when testing surveys that use these scripts you want to make sure that you receive any JavaScript errors that might occur. In Netscape and FireFox this just requires typing "JavaScript:" into the location bar of the main window. In IE, this requires changing the browser settings: Click tools. Click internet options. Click advanced. Uncheck the box labeled "Disable Script Debugging" and check the box labeled "Display a notification about every script error." For other browsers, please refer to the documentation for your browser.

The JavaScripts currently provided are as follows:

initial81.js

cfmc_ws81.js

cfmc_tmpl81.js

user_settings81.jsimg_81.js

dbr81.js

dragdrop.js

slider.js

placefocus.js

Cfmc_tmpl81.js

Cfmc_tmpl81.js is a the latest addition to the Scripts provided with the software. It groups together the template functions.

These scripts must be called in a certain order:

In the index page: initial81.js, then user_setting81.js; in header.tmpl/pagetop.tmpl: cfmc_ws81.js, cfmc_tmpl81.js and finally user_settings81.js.

You can add your own custom Script to CfMC's. The custom script must be called after user_settings81.js. This is explained further in the "Adding custom scripts" section.

Img_81.js, dbr81.js, dragdrop.js, slider.js, placefocus.js are examples of "custom" scripts. They are explained in the "Adding custom scripts" section.

NOTE: DO NOT COMBINE 8.0 SCRIPTS WITH PREVIOUS SCRIPT VERSIONS OR CFMC SOFTWARE RELEASED PRIOR TO 7.7

REMINDER: UNIX is case-sensitive, so make sure that the following JavaScript files do not contain any upper-case letters.

User_settings81.js

This script contains all of the customizable settings for the JavaScripts CfMC provides. You call it into the page after any other scripts you are using. If this script does not exist or cannot be read, then default settings will be used.

Place a call to user_settings81.js both in the starting html file after the call to initial81.js and in header.tmpl after the call to cfmc_ws81.js.

In the starting html file:

<script src="/cfmcweb/js/initial81.js" type="text/javascript"></script>

user_settings.jsin pagetop.htmlcfmc_ws.js

<script src="/cfmcweb/js/cfmc_tmpl.js" type="text/javascript"></script><script src="/cfmcweb/js/user_settings81.js" type="text/javascript"></script>

The script has comments for each section explaining what that section applies to. For more information on the individual settings please look at the details for that section or function.

Initial81.js

A call to initial81.js is placed in the starting html file for the survey along with a separate control script (user_settings81.js). Initial81.js allows for the passing of a password and username from a hotlink or a cookie. It opens a pop-up page and allows the page to be auto-submitted if there is a username and password provided. It will also automatically pass any data included in the URL string on to the survey.

There is a call to the script at the bottom of the file as well as to the new control script:

```
<script type="text/javascript" src="/cfmcweb/js/initial81.js"></script>
<script type="text/javascript" src="/cfmcweb/js/user_settings81.js"></script>
```

You control the settings for the script with user_settings81.js. This script can be set to defaults you wish to use for all studies or copied into the Web area's study directory and customized for a given job.

The use of the controls is explained below.

var login_from = "cookie:link" This provides the order in which the script looks for a username and password. If more than one setting is used, they process in the order given here. Also if more than one setting is used, each setting should be separated by a colon(:). The allowable settings are:

cookie: take username and password from a cookie if one exists and the study name in the cookie matches the studyname in the index page.

link: take the username and password from the URL. This is by default formatted as:

name=username and password=password if using both or id=password if using the generic username "Respondent." You can now customize these name-value pairs in user_settings81.js.

random: Should only be used on open studies. This will randomly generate username and password if they do not exist.

In the example above, if a cookie was set, the username and password would come from the cookie. If not, it would come from a hotlink. If neither existed, nothing would be filled in.

The relevant script section is the following:

```
var name_in_link = "name";
var password_in_link = "password";


var default_name = "respondent" //generic username in fone file
var id_in_link = "id";
```

If the link you send out contains only one name-value pair, initial81.js will assume you are using a generic value for the name in the fone file, such as "Respondent", so it will fill in the name input with the value of the javascript variable default_name

Therefore, a default link can be:

```
http://somedomain.com/somestudy/index.html?name="Name"&password="password"
```

Or if the name input value is equal to "Respondent" then it can be:

http://somedomain.com/somestudy/index.html?id="password"

If you change the defaults a link can be:

http://somedomain.com/somestudy/index.html?user="Name"&pin="password"

or

http://somedomain.com/somestudy/index.html?pid="password".

In the first case you have:

var name_in_link = "user";

var password_in_link = "pin";

In the second case you can have:

var default_name = "participant" //generic username in fone file

var id_in_link = "pid";

In this case, the default_name value does not show in the link but the name input will be filled in with "participant" (you will have "participant" in position 71.20 in the fone file for every record).

If the name and password values are filled from the link and you do customize those variables, then the defaults will be overridden. So, it is important to check which user_settings81.js is being used for the study.

var user_use_autostart = false; (true or false). This controls whether the respondent must click a "Submit" button to enter the survey or whether the index page will automatically be submitted after it is filled out by either a cookie or a hotlink.

var use_cookie = false; (true or false). This can be set to true if you wish to use cookies to control access to the survey. If it is true, then a cookie that contains username, password and study will be written to the respondent's machine. The cookie will be checked anytime a respondent tries to access the survey. The use of cookies is not a fool-proof way to control survey access. So, it is recommended that you use them on open surveys.

var cookie_lifetime = 30; (in days). This controls how long a cookie will remain valid.

var use_popwindows = true; (true or false). This controls whether the survey will be started in a new pop-up window or whether it will continue in the same browser window as index.html. It's generally a good idea to use pop-up windows to control respondents' use of their browser buttons. There are two methods provided for handling the formatting of the pop-up window. It can open to a fixed size or the JavaScript can determine what size the available window should be, based upon the respondent's screen resolution.

var adjwidth = 10; This adjusts the width of the expandable pop-up window by # of pixels. The window will be created # pixels less than the full size of the screen.

var adjheight= (screen.height - screen.availHeight) + 20; This adjusts the window height by the stated number of pixels. Change the 20 to some other pixel value to change the height of the pop-up window.

var xcorner = 0; (in pixels). This sets how many pixels from the left side of the screen the pop-up will be placed.

var ycorner = 0; - This sets how many pixels from the top of the screen the pop-up will be placed.

var xsize = screen.width - adjwidth; This sets how many pixels wide the pop-up window is, based on the adjwidth variable above.

var ysize = screen.availHeight - adjheight; This sets how many pixels high the pop-up window is set, based on the adjheight variable.

var DisableButtonsOnSubmit = false; (true or false) This can be set to true to disable (hide) the buttons once they've been clicked to submit the page.

For webCATI

You can pass values to fill in a webCATI index as well. Once the values are filled in from a link, then they cannot be edited. The only exception is when you are in practice mode. That can be changed at any time. None of these variables are required, and if they are not passed, then they will need to be typed in. If the input for a variable is not on the page, then it will be ignored.

The variables to pass in the link, separated by an & (ampersand) are ...

booth=	
practice_mode=	1=true, 2 or missing=false
special_types=	
extn=	
timezone=	
soundchannel=	

The link would look something like this:

http://yoururl/yourpath/index.html?booth=1&practice_mode=1&special_types=1234&extn=1&timezone=4&sound_channel=1

Cfmc_ws81.js

This script is designed to handle all JavaScript functions used inside webSurvent and the tmpl files.

It has numerous functions for dynamic data checking on a page created by webSurvent. It has a function for reading data in from a URL and storing it in hidden questions on the first screen of a survey. It has various functions for handling the tmpl files, such as a statusbar function for creating an image-driven status bar on screen, closing pop-up windows, changing the location of the main window once a survey has completed and functions for stopping the use of right-click and the browser movement buttons while in a survey. It also has CSS functions that enable you to dynamically change the CSS of the elements on screen.

In order to use the script, place a call to it in header.tmpl. It must be at the top of any webSurvent page you wish to use it on. For most functions to work, you need to call the specific functions you wish to use on any given page.

The current data checking functions are listed and described below. As requested, more functions may be added at a later date. For full examples of these scripts please visit our Web site and view the webSurvent 8.1 example demo.

The data checking functions

setcheckbox_to_radio – makes checkbox questions behave like single-response questions.

setdep – Allows you to set up conditional questions on the same page. It will not remove or present a question based on a condition and it will not allow a question to be answered unless the condition you defined is met. If a respondent tries to answer a question and the condition is not true, then they are presented with a user-defined message. If a respondent was at first allowed to answer a question and they change the question the condition was based on, then any responses given will be cleared. You can set up multiple conditions on the same page, but be careful to make sure that they are not mutually exclusive.

setmultiple – Checks for a minimum and maximum number of responses in a FLD/CAT type question.

setna – Allows you to set up a relationship between a single question and other questions on the page so that if an answer is given to the single question, then none of the other questions may be answered and vice versa.

setnodupes – Allows you to set up a relationship between a series of questions so that duplicate responses are not allowed. You can set an optional exception code that will be ignored when checking for duplicates. You can set a minimum number of questions that must be answered. You can set a maximum number of questions allowed to be answered. In this case if an exception code is defined then it will not count towards that maximum number of responses.

setnoskip – Allows you to force a response to some questions on a page while allowing others to be left blank. If a respondent submits the page and a required question is not answered, an error message is presented and they are taken to that question. This process will continue until all required questions on the page are answered.

setnum - Allows you to force numeric entries to be within a range (minimum and maximum) and to have the correct number of decimals. It will also prevent clicking on more than one exception code and clicking an exception code while entering a number. If a number is entered that is out of range or does not have the correct number of decimals entered, then an error is presented to the respondent and the entry is cleared.

setorder – Allows you to record the order of mention of a multi-response question. The order of mention gets stored into a hidden question. Exclusive response codes are allowed.

setother – Allows you to set up a relationship between a CAT/FLD question and any number of other-specify questions (either text or var type questions are allowed). It allows you to force the specify if an other is checked, allows you to use a pop-up other, places the cursor in the specify entry box if other is clicked. It will also clear the specify question if the other button is cleared.

SetOtherInterval – Allows you to set up a relationship between a CAT/FLD or NUM question and any number of other-specify questions (either text or var type questions are allowed); the driving question is linked to the other specifys on condition. It allows you to force the specify if an other value falls within a given interval, to use a pop-up other, to hide the other specify and places the cursor in the specify entry box if other is clicked. It will also clear the specify question if the other button is cleared.

setrank – Allows you to set a relationship between several ranking questions on a page so that each question must be given a unique rank. If a respondent attempts to give the same ranking to more than one question, then an error messages is presented and their last response is cleared. You can set up an exception code for each question that will be ignored when comparing ranks.

SetReveal – Allows you to hide and reveal portions of the page according to the current values of a base question. If the question is answered in the appropriate way then that area of the page will be revealed. If it is not then that area of the page will be hidden or remain hidden as the case may be and those inputs will be cleared. When the page loads if data exists then the checks are evaluated and the appropriate areas are shown. The base question can be a FLD, CAT or NUM. The other questions can be of any type. You can build as many of these relationships as you want on as many values as you want.

settab – Allows you to set a tab order on the page so that if a respondent tabs through questions it is done in a specific order. By default, browsers set the tab order based on the order of the element on the page. This function will not work with CAT or FLD type questions.

settotal – Allows you to set up a constant sum question as well as setting up a constant sum on Excel-type table structures. This links a series of numeric questions and provides a running total and a running grand total. The grand total is calculated off the table totals on rows and columns. You can force the total and the grand total to be specific target numbers if you choose. You cannot currently provide the target as a range; it must be a specific number. The script also checks to ensure that a valid number is entered. You can have either multiple calls to this function on the same page and pass the same question label to two or more different function calls (i.e. a question label can at the same time be part of a row and a column on the table), or you can simply have multiple calls with each call made to a different set of questions.

setunique – Allows you to define up to five responses as unique so that they cannot be given with any other response. You can also link this to a specify question. You have the option as to how a respondent will be presented the error. You can simply alert them that they have given a unique response and clear the last response given or you can ask them which response they wish to keep - the unique response or the other response(s). This function will only work with CAT or FLD type questions or VAR/TEXT questions for the specify.

textlen – Allows you to count the amount of text entered into a var or text type question. You can provide a warning message to the respondent when they begin to run out of space for the question. You can define exactly what that warning level is or you can choose to provide no warning at all. You can also prevent the entering of more text that you wish to allow. You can show the characters remaining if you create an input for it and tell the function that id of that input.

textex – Allows you to link a text question with a CAT or FLD type question that contains exception codes. You can choose to force a respondent to either enter text or an exception code if you choose. If a respondent enters text and chooses an exception code an error message is presented telling them they have already answered the question and the last response given is cleared. It will also prevent clicking on more than one exception codes.

CSS functions

SetStyle – Allows you to change the specific style of an object. You can change more than one style element with one function call.

SetClass – Allows you to change the class of an object

ChangeStyleRules – Allows you to change specific style elements of a class

InsertText – Allows you to dynamically write text into a designated element on the page

Calling the functions

In a display question, place a JavaScript to call the function or functions you wish to use on that particular screen/page before the endgrid. This call must always be at the bottom of the page after the last question to be presented. You can call multiple functions on the same page. Be careful when doing this because some functions may not be compatible with each other, particularly if they are being used on the same question or response.

Setother function

Purpose: To link a CAT or FLD question with up to five other-specify responses to VAR/TEX questions to store the specify.

Three control options allow you to force the other specify always to be entered if other is chosen, to use a pop-up other or not and to hide the other specify when the other specify response is not clicked.

Usage: Place a call to the function in a display question at the bottom of the grid. Passing the correct parameters. You can use this function multiple times on the same page.

```
<script type="text/javascript">
setother(forceo,popit,hideother,qlabel,other1,otherN);
</script>
```

The parameters to pass are:

Force = Value is true or false. True forces the entry of the other specify. False does not

Popit = Value is true or false. True generates a pop-up window for the entry of the specify. False does not

Hideother=Value is true or false. True hides the other box when the other code is not selected and shows the box when the other code is selected.. False always shows the other box.

Qlabel = The label of the question with the other button/check box

Other1=contains the value of the first other code and the label of the relative var/text question to store the specify.

OtherN=contains the value of the N-th other code and the label of the relative var/text question to store the specify

All parameters are required and should be enclosed in quotes and separated by commas. You should at least have one var/text specify question label.

```
<script type="text/javascript">
setother("true","false","true","Q1","5:Q1_OTH",6:Q2_OTH","7:Q3_OTH,"8:Q4_OTH","9_Q5_O
TH") ;
</script>
```

The function call above would create a link between the !cat/!fld question q1 response value "5" and q1_oth, a var/text type question, question q1 response value "6" and q2_oth, a var/text type question, question q1 response value "7" and q31_oth, a var/text type question and so on. The specify would be forced if other was chosen and no pop-up would be used. The other specify boxes will be hidden unless the other codes are selected.

Another example is:

```
<script type="text/javascript">
setother("true","false","false","Q1","5:Q1_OTH",6:Q2_OTH","7:Q3_OTH") ;
</script>
```

The function call above would create a link between codes 5,6,7 of the !cat/!fld question q1 and the three specify Q1_OTH, Q2_OTH AND Q3_OTH.

SetOtherInterval function

Purpose: To link a !cat/!fld (single or multi-response) or a !num question with any number of other specify responses to !var/!text questions to store the specify. Input in the other specify questions is only allowed when the value(s) of the driving question fall(s) within a declared interval. The interval for every other specify is defined with two values: low and high.

Three control options allow you to force the other specify always to be entered if other is chosen, to use a “pop-up” other and to hide the other specify when the other specify response value does not fall within the relative interval.

Usage: Place a call to the function in a display question at the bottom of the grid, passing the correct parameters. You can use this function multiple times on the same page.

```
<script type="text/javascript">
SetOtherInterval(ForceOther,PopOther,HideOther,Type,QuestionLabel,Other1,Othern);
</script>
```

The parameters to pass are:

ForceOther = Value is true or false. True forces the entry of the other specify. False does not.

PopOther = Value is true or false. True generates a pop-up window for the entry of the specify. False does not. If PopOther is “true”, then it is recommended to use !var,h for the other specify.

HideOther=Value is true or false. True hides the other box when the other code does not fall within the declared interval and shows the box when the other code falls within the interval. False always shows the other box. This option works best with the survent popup other specify.

QuestionLabel = The label of the question with the other button/check box

Other1=contains the low and high values for the first interval, and the label of the relative var/text question to store the specify.

Othern=contains the low and high values for the n-th interval, and the label of the relative var/text question to store the specify.

All parameters are required and should be enclosed in quotes and separated by commas. You should at least have one var/text specify question label.

```
<script type="text/javascript">
SetOtherInterval("true","false","true","num","Q1","1:2:Q11OTH","11:15:Q12OTH");
SetOtherInterval("true","false","false","fld","Q1A","1:3:Q1A1OTH","5:5:Q1A2OTH");
</script>
```

The first function call above would create a link between the !num question Q1 and the other specify questions Q11OTH and Q12OTH. Input in Q11OTH is allowed only when Q1 >= 1 and Q1 <= 2 and input in Q12OTH is allowed only when Q1>= 11 and Q1 <= 15; the other specify do not pop up and are hidden.

The second function call above would create a link between the !fld question Q1A and the other specify questions Q1A1OTH and Q1A2OTH. Input in Q1A1OTH is allowed only when Q1A >= 1 and Q1A <= 3 and input in Q1A2OTH is allowed only when Q1A = 5; the other specifys do not pop up and are not hidden.

Settotal function

Purpose: To create a link between a series of numeric questions and allow sums on an excel type table structure. To show a running total and a grand total (if specified) as the numbers are entered into the page and if set force the total and the grand total to equal a specific target and grand target.

Usage: Place a call to the function in a display question at the bottom of the grid, passing the correct parameters. You can use this function multiple times on the same page.

```
<script type="text/javascript">

function settotal(eltype, force_total, force_grand_total, prefill, num_decs, labels, totalfld, target, grand_total, grand_target)

</script>
```

The parameters to pass are:

Eltype = Value is row or column. This specifies whether you are calling the function on a row or a column.

Force_total = Value is true or false. True forces the total to equal the target. False does not.

Force_grand_total = Value is true or false. True forces the total to equal the target. False does not. If you do not have a grand total specified, you still need to pass an empty value (“”).

Prefill = Value is true or false. True prefills a 0 in each question when the page is first loaded. False does not.

Numdecs = The number of decimals allowed

Labels = The question labels of the numeric questions to count towards the total

Totalfld = The id of the input tag where the running total will be displayed

Target = The target value. The value to check the total against

Grand_total = The id of the input tag where the running grand total will be displayed

Grand_target = The grand total target value. The value to check the grand total against

All parameters are required and should be enclosed in quotes and separated by commas. If a parameter is not needed, it can be passed as "". In the case of the labels argument, the entire string should be in quotes and each question label should be separated by a comma.

```
<script type="text/javascript">

settotal("col", "true", "true", "true", 0, "SA_1,SB_1,SC_1,SD_1", "TOTAL9",  "80" , "TOTAL12","320");

</script>
```

The function call above would create a link between questions SA_1-SD_1, placed along a column. Whenever a number is entered, it will be added to the running total and displayed in the TOTAL_9 input tag.

Since force_total is set to true, if the numbers exceed the target value of 80, then a warning will be displayed and the last number entered will be cleared; the background colour of the total tag will change and switch back to its original value after acknowledgement of the warning. You can set this background colour editing the value of var cs_error_background_color in user_settings81.js.

Since force_grand_total is set to true, if the sum of the totals exceeds 320, then a warning will be displayed and the last number entered will be cleared; the background of the grand total tag will change as described above for the total tag. You can set the background color in the same way as described above.

Since prefill is true, a 0 would be written into each question when the page is first loaded.

Additionally, when the page is submitted, the total and grand total values will be checked against the respective targets and, if they are not equal, the submission of the page will be stopped and an error warning will be displayed.

```
settotal("row", "true", "true", "true", 0, "SB_1,SB_2,SB_3", "TOTAL2",  "40","TOTAL12","320");
```

The function call above would follow the same rules as the previous one, except that the question labels are along a row. In this case, the common question label between the two calls is SB_1.

Setdep function

Purpose: To create on-screen skip patterns so that designated questions cannot be answered if a question on the same page does not have the proper response. This allows for a one-to-many skip pattern (all questions on a given page can be based on a single question on the same page). For a many-to-one relationship see the setna function.

Usage: Place a call to the function in a display question at the bottom of the grid; passing the correct parameters. You can use this function multiple times on the same page.

```
<script type="text/javascript">
setdep(slabel,sval,srel,olabel,msg);
</script>
```

The parameters to pass are:

slabel: subject question: The question label of the question the condition is based on value:

sval: The value to check for

srel: The conditional operator to use.

- eq for equal
- ne for not equal
- ge for greater than or equal
- le for less than or equal
- gt for greater than
- lt for less than

olabel: object question(s): The question that should not be answered if the condition is not true. (If there are multiple questions separate each label with a comma and place the whole string of labels in quotes.)

Msg: error message: The message you want to display if the subject question(s) is answered and the condition is not true.

All are required. Each argument must be in quotes and the arguments must be separated by commas.

```
<script type="text/javascript">

setdep("q1","0","ne","q6","You may not answer this question if you answered yes above");
setdep("q2","1","eq","q4,q5,q7","You may not answer this question if you answered yes above");

</script>
```

The first function call above would create a link between the question q1 and q6 so that q6 cannot be answered if q1 = 0. The second function call would create a relationship between q2 and q4, q5 and q7 so that they could only be answered if the response to q2 was 1.

SetReveal function

Purpose: To hide and reveal portions of the page according to the current values of a base question.

Usage: Place a call to the function in a display question at the bottom of the grid; passing the correct parameters. You can use this function multiple times on the same page.

Take an input or set of inputs and wrap them in a div or span with a specific ID. Tell SetReveal what the base question is, a value, a relationship (eq, gt, etc...), the id of the div or span you created and the name of the input questions inside that container.

If the question is answered in the appropriate way then that area of the page will be revealed. If it is not, then that area of the page will be hidden or remain hidden as the case may be and those inputs will be cleared. When the page loads if data exists then the checks are evaluated and the appropriate areas are shown.

The base question can be a fld, cat or num. The other questions can be of any type. You can build as many of these relationships as you want on as many values as you want.

You can have multiple values to check against. For example:

```
"Q3","1:ne:show_num1:Q1","2:eq:show_num2:Q2"
```

You can have multiple data elements, such as

```
"Q3","1:ne:show_Q4:Q4:Q5"
```

The function is layered. So if you clear data from an item and that item has other questions based on it, they are cleared as well.

```
<script type="text/javascript">
SetReveal("base question","value:condition:show element:data element(s)");
</script>
```

The parameters to pass are:

base question: the question label of the driving question.

value: the value of the base question to be checked

condition: one of eq (equal), ne (not equal), gt (greater than), lt (less than) ge (greather than or equal to), le (less than or equal to).

show element: the id of the element to be shown/revealed.

data element(s): the labels of the questions contained in the divs.

```
<script type="text/javascript">
SetReveal("Q1","10:gt:show_radio:Q3");
SetReveal("Q3","1:ne:show_num1:Q1","2:eq:show_num2:Q2");
</script>
```

The first function call above would reveal Q3, wrapped by the div of id show_radio, when Q1 > 10.

The second function call above would reveal Q1, wrapped by the div of id show_num1, when the value of Q3 is <> 1 and would reveal Q2, wrapped by the div of id show_num2, when Q3 =2

```
<script type="text/javascript">
SetReveal("Q3","1:ne:show_Q4:Q4:Q5");
</script>
```

The first function call above would reveal Q4 and Q5, wrapped by the div of id show_Q4, when Q3 <> 1.

Setnodupes function

Purpose: To disallow duplicate responses across a series of questions. You can optionally set an exception code that will not be considered a duplicate response. You can also set a minimum and a maximum number of responses. Multiple response questions are disallowed.

Usage: Place a call to the function in a display question at the bottom of the grid; passing the correct parameters. You can use this function multiple times on the same page. You will need to write the function call for each set of questions you wish to check.

```
<script type="text/javascript">

setnodupes(qlabels,exception,minimum,maximum);

</script>
```

The parameters to pass are:

qlabel: the labels of the questions to check, separated by a comma

exception: the exception code to be ignored in the check

minimum: the minimum number of responses allowed

maximum: the maximum number of responses allowed

The qlabels are required. Exception, minimum and maximum can be left blank. Each argument must be in quotes and the arguments must be separated by commas.

```
<script type="text/javascript">

setnodupes("q1:q2:Q3:Q4:Q5","","","");

</script>
```

Would check questions q1 to q5 for duplicate responses. No exceptions code and no check for minimum or maximum.

```
<script type="text/javascript">

setnodupes("q1:q2:Q3:Q4:Q5","5","","");

</script>
```

Would allow response 5 in all questions as it is defined as the exception code in this case.

```
<script type="text/javascript">

setnodupes("q1:q2:Q3:Q4:Q5","5","2","4");

</script>
```

Would require that at least two responses were given and that no more than four valid responses (excluding exception codes) were given.

Setnoskip function

Purpose: To force a response to some questions on a screen while allowing others to remain blank.

Usage: Place a call to the function in a display question at the bottom of the grid; passing the correct parameters. You can use this function multiple times on the same page. You will need to write the function call for each question you wish to force.

```
<script type="text/javascript">

setnoskip(qlabel,msg,confirmation);

</script>
```

The parameters to pass are:

qlabel: the label of the question you want to force

Msg: error message: The message you want to display if the question is not answered when the page is submitted.

Confirmation: true or false. If set to true and the question is not answered when the page is submitted, an alert is displayed and the respondent has to provide an answer to the question; if set to false a confirmation dialogue is displayed asking whether to continue and not answer the question or not continue and answer the question.

All are required. Each argument must be in quotes and the arguments must be separated by commas.

```
<script type="text/javascript">

setnoskip("q1","Please answer q1");

</script>
```

Hint: In order to place focus on a fld/cat question that has not been answered, you can place a link in the text of the question. For example:

```
<a href="" name="Q1"></a>
```

For var, num or text questions the cursor will be placed into the question that was not answered.

The first function call above would require that q1 must be answered when the page is submitted. If it is not answered, then the message "Please answer Q1” would be displayed, and, if possible, the cursor or page focus would be placed at q1.

Setnum function

Purpose: To check numeric entries for the correct range and number of decimals. It checks to see if a numeric entry is in the correct range and has the correct number of decimals. It also disallows entering a number and clicking an exception code.

Usage: Place a call to the function in a display question at the bottom of the grid; passing the correct parameters. You can use this function multiple times on the same page. You will need to write the function call for each question you wish to check.

```
<script type="text/javascript">
setnum(qlabel,qmin,qmax,numdecs);
</script>
```

The parameters to pass are:

qlabel:	The label of the numeric question to check
minimum:	The minimum value allowed
maximum:	The maximum value allowed
numdecs:	The number of decimals required

All are required. Each argument must be in quotes, and the arguments must be separated by commas.

```
<script type="text/javascript">
setnum("Q25A","0","100","0");
setnum("Q25B","0","100","2");
</script>
```

The first function call above would set Q25a with a minimum value of 0, a maximum of value of 100 and the allowed number of decimals at 0. If there are exception codes associated with Q25A, it will automatically disallow answering more than one exception code and answering an exception code as well as entering a numeric value.

Setorder function

Purpose: to record the order of mention of responses in a multi-response question. Selecting an (optional) exception code will clear the other responses

Usage: Place a call to the function in a display question at the bottom of the grid.

Passing the correct parameters. You can use this function multiple times on the same page. You will need to write

The function call for each question you wish to check.

```
<script type="text/javascript">

setorder (qlab,hidlab,maxr,excs)

</script>
```

The parameters to pass are:

qlab: the question label you want to store the order of mention for.

hidlab: the var,h label to store the codes values in

maxr: the max number of responses in the FLD/CAT question.

excs: the exclusive codes, optional. Selecting an exclusive code will clear the other responses.

If defining multiple exclusive codes, separate each with a colon.

All the parameters are required except excs. Each argument must be in quotes and the arguments must be separated by commas.

```
<script type="text/javascript">

setorder("qset1","qorder1","4");

setorder("qset2","qorder2","4","8:9");

</script>
```

Setrank function

Purpose: To check a series of ranking questions to make sure the same rank cannot be given. The Questions can be numeric, or fld/cat with buttons or dropdown boxes. You can also specify the number of items to rank. If the exception code is specified and we specify the number of items to rank, every item must be given either a rank or an exception code.

Usage: Place a call to the function in a display question at the bottom of the grid, passing the correct parameters. You can use this function multiple times on the same page. You will need to write the function call for each set of questions you wish to check.

```
<script type="text/javascript">
setrank(rlabels,exceptcode,ritems);
</script>
```

The parameters to pass are:

rlabels: The labels of the questions to check. Each label separated by commas.

exceptcode: An exception code that can be answered for all questions

ritems: the max number of items to be ranked

Exceptcode is optional and should not be specified for numeric questions. Ritems can be specified; if you want to have all the items ranked, pass a value equal to the number of items. If ritems is not specified, the check for the number of items will not be performed. Each argument must be in quotes and the arguments must be separated by commas.

```
<script type="text/javascript">
setrank("R2A,R2B,R2C,R2D,R2E","","5");
setrank("R3A,R3B,R3C,R3D,R3E","9","5");
setrank("R4A,R4B,R4C,R4D,R4E","","3");
setrank("R5A,R5B,R5C,R5D,R5E","9","3");
</script>
```

The first call will check that all the items have a rank. The second call will check that every item has either a rank or an exception code. The third call will ensure that three items will be ranked. The fourth call will check that max three items have a rank and all the items are either ranked or have an exception code.

Numeric questions that have exception codes will behave the same way as if the exception code was passed to setrank.

Settab function

Purpose: To create a specific tab order onscreen so that if the respondent hits the tab key, they move through the page in a specific order. You can only set tab order for NUM, VAR or TEXT type questions.

Usage: Place a call to the function in a display question at the bottom of the grid. There can only be one call to settab on any given page.

```
<script type="text/javascript">
settab(tablabels,setcursor);
</script>
```

The parameters to pass are:

tablabels: The labels of the questions to set tab order for (in the order you want them tabbed through).

setcursor: If true, it will place the cursor in the first question in the tab order. If false, it will not.

```
<script type="text/javascript">
settab("Q9_1,Q9_2,Q9_4,Q9_5,Q9_3","true");
</script>
```

The function call above would set the tab order onscreen to be q91, q9-2, q9_4, q9_5, q9_3. In this case, the cursor would be placed in q9_1, if possible.

Setunique function

Purpose: To designate a given response on a multiple-response FLD/CAT question as a unique response. To prevent a respondent from giving another answer and a unique response.

If this is set to false, the respondent is told they can answer a unique response and anything else. It clears the last response given.

Usage: You place a call to the function in a display question at the bottom of the grid. You can use this function multiple times on the same page.

```
<script type="text/javascript">

setunique(showalert,qlabel,olabel,unique1,unique2,unique3,unique4,unique5)

</script>
```

If this is set to false, the respondent is told that they can answer a unique response and anything else, and it clears the last response given. If set to true, it prompts the respondent to choose what should be maintainted (the unique response or the other response(s).

The parameters to pass are:

showalert : true or alert - alerts the respondent and clears the last value checked. False or confirm – presents a confirmation messages and asks the respondent to choose which response(s) to keep. None - presents no message to the respondent and just clears values appropriately: for example, if the last item clicked is a unique response all other responses are cleared; if the last item checked is not a unique response then any unique responses are cleared.

qlabel: This is the label of the question to check.

olabel: The labels of the other specify box if there is one. You can have up to five other specify boxes. Leave blank if there is not one associated with this question.

exception code 1-5 (you can define up to five exception codes). The format is response code: response text. Each exception listing should be in quotes followed by a comma.

The first three arguments are required. Only define the exception code arguments for the ones you need to use. Each argument must be in quotes and the arguments must be separated by commas.

```
<script type="text/javascript">

setunique("true","Q28A","","8:Not interested in any items");

setunique("false","Q28B","Q28BOTH: Q28BOTH2 ","98:Not interested in any items","99:Decline to state");

</script>
```

The first function call above would make response 8 for q28a a unique response. If a respondent gave another response, an alert would be displayed and the last response would be cleared. The second function call makes responses 98 and 99 unique responses and also includes two other-specify responses. If a unique response and some other response is given, then the respondent is prompted as to which response(s) should be kept.

Textlen function

Purpose: To count how many characters are entered into a VAR or TEXT question. It will give a warning when the maximum is approached and/or stop the respondent from entering more text than is allowed.

Usage: Place a call to the function in a display question at the bottom of the grid and passing the correct parameters. You can use this function multiple times on the same page.

```
<script type="text/javascript">
textlen(txlabel,maxcol,warncol,mincol)
</script>
```

The parameters to pass are:

txlabel: The label of the question to check (must be text or var)

maxcol: The maximum number of characters allowed

warncol: Presents a warning at maximum minus this number. If 0, then no warning is displayed

mincol: The minimum of characters allowed

txcounter: The label of an input tag to show the characters remaining to be typed

All parameters are required and should be enclosed in quotes and separated by commas.

```
<script type="text/javascript">
textlen("Q8a","200","100",”10”,””);
textlen("Q9a","100","0",”0”,””);
textlen(“Q10”,”100”,”0”,”0”,”Q10COUNT”);
</script>
```

The first function call sets q8a to a maximum number of 200 characters and a minimum of 10 and sets a warning display when the respondent is 100 characters short of the maximum and an alert error message upon submit if the characters are less than 10. The second function call sets only a maximum of 100 characters for q9a and no warning will be displayed. In both cases, any characters entered over the maximum will not be saved. The third function call allows for 100 characters inQ10 with no minimum and no warning. It will show the remaining characters to be typed in an input tag called Q10COUNT.

Textex function

Purpose: Associate a text question with a FLD/CAT type question so that you can only answer one or the other. It allows for the setup of exception codes to text questions.

Usage: To place a call to the function in a display question at the bottom of the grid; passing the correct parameters. You can use this function multiple times on the same page.

```
<script type="text/javascript">
textex(txtabel,exlabel,message,force)
</script>
```

The parameters to pass are:

txlabel: The label of the text question

exlabel: The label of the exception question

message: Text inserted into error message if both text and exception code are answered; or forcing answer and no answer is given.

force: If true, an answer to one or the other question is required. If false, you can leave both blank.

All parameters are required and should be enclosed in quotes and separated by commas.

```
<script type="text/javascript">
textex("Q27A","Q27AREF","Question 1","true");
textex("Q27B","Q27BREF","Question 2","false");
</script>
```

The first function call above sets a relationship between q27a and q27aref so that only one of them can be answered. It also sets a check so that a response is required for one or the other when the page is submitted. The second function call sets the same relationship between q27b and q27bref, but it allows them to be blank when the page is submitted.

Setna function

Purpose: To create a one-to-many relationship between questions so that if one is answered, then the others cannot be. Or, if any of the others are answered, then the one cannot be.

Usage: To place a call to the function in a display question at the bottom of the grid; passing the correct parameters. You can use this function multiple times on the same page.

```
<script type="text/javascript">
setna(subject question,object questions,text for subject,text for object questions);
</script>
```

The parameters to pass are:

Subject question: The label of the single question

Object question: The label(s) of the object (many) questions. Each label should be separated by a comma.

Text for subject: This is the text used in regard to the subject (single) question if an error occurs

Text for objects: This is the text used in regard to the object question(s) if an error occurs. Again, the individual question text should be separated by commas. The order should be the same as in the object question argument above.

All parameters are required and should be enclosed in quotes and separated by commas.

```
<script type="text/javascript">
setna("q7","q1,q2,q2oth,q3","Question 7","Question 1,Question 2,Question 2,Question 3");
setna("q8","q10,q11,q12,q13","Question 3,Question 4,Question 6,Question 7");
</script>
```

The first function call above sets a relationship between q7 and q1,q2,q2oth and q3 so that q7 cannot be answered if there is a response to q1, q2 q2oth or q3 and vice-versa.

setcheckbox_to_radio

Purpose: To make checkbox questions behave as single response questions.

Usage: To place a call to the function in a display question at the bottom of the grid, passing the correct parameters. You can use this function multiple times on the same page.

```
<script type="text/javascript">
setcheckbox_to_radio(qlabel)
</script>
```

The parameters to pass are:

Qlabel: the label of the question with checkboxes you want to behave as single response question.

The parameter should be enclosed in quotes.

```
<script type="text/javascript">
setcheckbox_to_radio("QA");
</script>
```

The first function call above makes QA behave like a single-response question.

setmultiple

Purpose: To check for a minimum and maximum number of responses in a multi-response FLD/CAT type question. Selection of an exception code will clear the other responses.

Usage: Place a call to the function in a display question at the bottom of the grid, passing the correct parameters. You can use this function multiple times on the same page.

```
<script type="text/javascript">

setmultiple(qlabel,min,max,exceptions)

</script>
```

The parameters to pass are:

Qlabel: the label of the question to check

Min: The minimum number of responses that can be accepted

Max: The maximum number of responses that can be accepted

Exceptions: the FLD/CAT exception codes

All arguments are required. Exception codes can be passed as "" if there are none; if there are more than one, then they have to be separated separate with ","

```
<script type="text/javascript">

setmultiple(“QA”,”2”,”4”,”8:9”);

</script>
```

The function call above will enforce a minimum of two responses and a maximum of four. If codes 8 or 9 are selected, the other responses will be cleared.

CSS functions

SetStyle

Purpose: To change a specific style of an object. Can change more than one style element with one function call.

Usage: Place a call to the function in a display question at the bottom of the grid, passing the correct parameters. You can use this function multiple times on the same page. You can also use this function in your own custom JavaScripts so that you can change the way a page looks based on some action taken by the user.

```
<script type="text/javascript">
SetStyle(element ID,prop1,prop2,propN)
</script>
```

The parameters to pass are:

Element: the id of the object whose style elements we want to set.

Prop1: name-value pair for the first property to be changed.

Prop2: name-value pair for the second property to be changed.

PropN: name-value pair for the n-th property to be changed.

All arguments are required. If there are more than one property name-value pairs, then they have to be separated separate with a comma.

```
<script type="text/javascript">
SetStyle("Q1",”borderStyle:dotted","borderColor:red");
</script>
```

The function call above would set the object’s border style to dotted and the border color to red.

SetClass

Purpose: To change the specific class of an object.

Usage: Place a call to the function in a display question at the bottom of the grid, passing the correct parameters. You can use this function multiple times on the same page. You can also use this function in your own custom javascripts. So that you can change the way a page looks based on some action taken by the user.

```
<script type="text/javascript">
SetClass (element ID,newclass)
</script>
```

The parameters to pass are:

Element: the id of the object whose class we want to change.

Newclass: the name of the new class we want to set the object to.

```
<script type="text/javascript">
SetClass ("Q1","boxed")
</script>
```

The function call above will set the element to the class “boxed”.

ChangeStyleRules

Purpose: to change specific style elements of a class.

Usage: Place a call to the function in a display question at the bottom of the grid, passing the correct parameters. You can use this function multiple times on the same page. The class is passed to the function by locating its position in the “rules” array (i.e. an external style sheet linked to the page). The ChangeStyleRules function will try to access the class in the first external style sheet defined. You can also use this function in your own custom JavaScripts so that you can change the way a page looks based on some action taken by the user.

```
<script type="text/javascript">

ChangeStyleRules(crules_array_num,prop1,prop2,propN)

</script>
```

The parameters to pass are:

crules_array_num: the position within the rule array of the class whose properties will be changed

Prop1: name-value pair for the first property to be changed.

Prop2: name-value pair for the second property to be changed.

PropN: name-value pair for the n-th property to be changed.

```
<script type="text/javascript">

ChangeStyleRules("2", "borderStyle:dotted","borderColor:red");

</script>
```

The function call above will set the class’ border style to dotted and the border color to red. The class is located in position 2 (3rd class) in the rules array.

InsertText

Purpose: Insert designated text and html into an element on the page. This function is used in the main script to implement the status bar, for example.

Usage: Place a call to the function in a display question at the bottom of the grid or into your own custom scripts, passing the correct parameters. You can use this function multiple times on the same page.

```
<script type="text/javascript">
InsertText(element ID, content);
</script>
```

The parameters to pass are:

element ID - The id of the object you want to write text into. Gener,y this will be a div, span or p tag.

content - The content to pass. This can be either a string of text or it can be built as a variable containing text, html and even the current value of certain elements

```
<script type="text/javascript">
var content = "text and html to be displayed";
InsertText("thisid", content);
</script>
```

The function call above would place "text and html to be displayed" into an html object named thisid. The ID must exist on the page and should not be a formatting element and not an input element.

Debugging cfmc_ws81.js

It is essential that when you use JavaScripts, you set your browser to display any script errors. Please review the documentation or help files for your browser to see how to do this. Once this is done, if you receive a message that says “Object Expected,” it means that either the cfmc_ws81.js script has not been loaded or that one of the function calls being used on the page is incorrect. To debug, first make sure that the script call is correct. If it appears to be correct, then view the source of the page and look at the line number provided with the above error message. This should at least make it clear which function call is the problem.

Programmer alerts. Programmer error messages are a part of the script. If you call a function with a question or a response code that does not appear on the page, you will be alerted and the function being called will stop. This may make it more difficult to use the script with pages that have if conditions on them but it prevents accidentally creating a JavaScript error for a respondent. See using the data checking function on conditional questions below.

Debugging the functions. There are two variables in user_settings81.js that allow you to turn various levels of debugging on. Var debug_value = # determines which area of the script you wish to debug. When set to 0, no debugging alerts will be presented. Generally, the settings are:

1 - for setup checks

2 - for onscreen checks

3 - for onsubmit checks

4 - for onload checks

The second variable is: var debug_function = “settotal”; Set debug_function specifies the function name you wish to debug. The options are: setother, settotal, setdep, setnoskip, setnum, setorder, setrank, settab, setunique, textlen, textex, setna, setcheckbox_to_radio and setmultiple

In combining these settings, you can view the script as it goes through the various processes used in a function. By looking at set-up checks, you can make sure that the arguments you provided are correct. By looking at check functions, you are making sure that the actual on-screen checks are occurring properly. By looking at onsubmit checks, you are making sure that checks upon submission of the page are occurring properly. By looking at onload checks, you are making sure that checks made when the page is loaded are occurring properly.

Some functions do not have all of these checks. So if you set a level for a function and it does not exist, nothing will be displayed. Finally, if turning debugging on, remember that the script in the JavaScript directory applies to all jobs, so you may want to copy it to your study directory and change the call to point there.

Using the data checking function on conditional questions

If a function uses a question that is not actually written to the screen a programmer alert will be displayed and the function will stop running. In order to avoid this, you need to call or build question labels conditionally, as well.

For functions that refer to a single question (such as setnoskip, setnum, textex, etc…) simply use the same condition used on the question itself. For example:

```
{q2: !if q1(<>1)
a numeric question
!num,,,0-99,,DK:"Don't Know"}
{!if q1(<>1)
<script type="text/javascript">
setnum("Q2","0","99","0");
</script>
!disp }
```

For functions that refer to multiple questions or a string of questions (settotal, setdep, setna, etc.), use the following build_qlist function which allows you to build dynamically a string containing the question labels to pass to a data checking function.

build_qlist

Purpose: To allow building dynamically a list of question labels to pass to a data checking function.

Usage: Place a call to the function in a display question at the bottom of the grid, passing the correct parameters, before the data checking function call. You should use this function multiple times on the same page.

```
<script type="text/javascript">
build_qlist(qlabel)
</script>
```

The parameters to pass are:

Qlabel: the label of the question to check. All arguments are required.

```
>repeat $a=1,2,3
{
!if Q22($a)
<script type="text/javascript">
build_qlist("Q22A_$a");
</script>
!disp}
>end_repeat

{
<script type="text/javascript">
setrank(qlist,"");
</script>
!disp}
```

The function calls above will build a list of the labels to pass to setrank, based on some previous conditions being met. If, for example, Q22 responses were 1 and 3, then the call to setrank(qlist,""); would be equivalent to setrank("Q22A_1,Q22A_3","");

Customizing error messages

Look in user_settings81.js for any message that can be changed. These should be clearly commented as to which message goes with what function as well as when the message is displayed. If a message is not found, then a default setting for that message will be used.

//** UNIQUE RESPONSE MESSAGES.

//msg1 comes up if using an alert (showalert is set to true)

//msg2 comes up if using a confirm (showalert is set to false)

var unique_msg1 = "You cannot answer _UNIQUERESP_ and some other response.";

var unique_msg2 = "You checked _UNIQUERESP_ and some other response. If you intended to answer _UNIQUERESP_ click 'OK' and all other responses will be cleared. If you did not intend to answer _UNIQUERESP_ click 'CANCEL' and that response will be cleared."

//** OTHER SPECIFY MESSAGES.

//msg1 comes up if using a pop-up other and it is the text used in the pop-up (popit is set to true).

//msg2 comes up if force is set to true, the other response is checked and no specify was entered.

var other_msg1 = "Please provide a specific response";

var other_msg2 = "You checked other and did not enter anything. Please provide a specific response for _QUESTIONNAME_";

//** CONSTANT SUM MESSAGES.

//msg1 if the entry is not a number or less than 0

//msg2 if incorrect number of decimals entered

//msg3 if number entered cause total to exceed target and force total is true

//msg4 comes up on submit if force is set to true and the total does not equal the target

Message rules:

1. There can be no double quotes in the messages.
2. Only one line per message is allowed.
3. Anything in upper case and surrounded by underscores (_UNIQUERESP_) is a place holder that will be filled in by the script. You can move them around as you choose, but DO NOT REMOVE THEM FROM THE MESSAGE. If you do, when the message is to be displayed, the respondent will receive a JavaScript error rather than the assigned message.
4. DO NOT change the variable names or you will break the function.

Multi-language error messages:

Create a copy of user_settings81.js for each language. Keep in mind that special character encoding must be done in Unicode (UTF8) in order to display properly.

Set an !html_define in your spec for the language being used:

```
{lang: hide
!fld
1 English
2 Spanish
3 French }
{!if lang(<>1)
!goto nextlan1}
{!html_define English}
{nextlan1: !if lang(<>2)
!goto nextlan2}
{!html_define Spanish}
{nextlan2: !if lang(<>3)
!goto donelang}
{!html_define French}
{donelang: !goto }
```

When you call user_settings81.js in your header.tmpl, use >ifdef to call in the correct version of the file.

```
>ifdef English
<script src="/cfmcweb/js/user_settings_english.js" type="text/javascript"></script>
>endif
>ifdef Spanish
<script src="/cfmcweb/js/user_settings_spanish.js" type="text/javascript"></script>
>endif
>ifdef French
<script src="/cfmcweb/js/user_settings_french.js" type="text/javascript"></script>
>endif
```

Call cfmc_ws81.js as you normally would.

Other functions

parse_query function:

Purpose: To read data that is passed from the index page into the first page of the survey and store it in hidden var,h questions. There are two methods for passing this data.

The first method uses a hidden input named USER_DATA that is placed on the index page. The value of this input can be anything you want and filled in any way you want. There is a limit of 120 characters. This information is accessible with JavaScript throughout the survey.

The second method uses information passed through the url itself. This is an automatic function of index.html so you only need to set the script up in your spec to gather and store this data. This method will only work on the first page of the survey but you can pass approximately 2000 columns of information (depending on the browser type).

Method 1: Storing information from USER_DATA

Usage: Insert the script at the bottom of the page (before the end grid). This script needs to be just the call to parse_query function. You'll need to pass the name of the var question to store data in to the function.

Index page setup: You need to place the following in the index page for the study.

```
<input type="hidden" name="USER_DATA" value="this is user data"/>
```

The question setup: The question used for storage must be a var type question. It does not need to be hidden using var type, h but it is recommended. Make sure the question is large enough to store the data coming from USER_DATA.

In the qpx:

```
{grid1:
!grid}

{lurk:
!var,h,40,0}

{
<script type="text/javascript"|
  parse_query("lurk");
</script>
!disp}

{!endgrid}
```

Method 2: Storing information from the URL

Usage: Insert a script at the bottom of the page (before the end grid). This script needs to contain two matched arrays: one for parsing the URL query string and one for the question labels to store the questions in, and a call to the parse_query function.

Index page setup: There is nothing extra needed in the index.html file.

The question setup: All questions used for storage must be !var type questions. They do not need to be hidden using VAR type,h, but it is recommended. Make sure the questions are large enough to store the data for each element from the URL.

```
{g_q1:
Welcome to the survey. We've gathered the information you passed and
it will be displayed on the next screen.
!grid,b}

{respname:
!var,h,40,0}

{age:
!var,h,2,0}

{gender:
!var,h,1,0}

{other:
!var,h,2,0}
```

The script setup: There are two arrays that must be created by the programmer: one for reading in data from the URL, the other to tell the script where to store each item of data. These arrays must have the same number of elements and the elements order should match. The first item in the URL array will write to the first question in the VAR array.

General array rules:

Each array element must be in quotes followed by a comma.

The last array element must be followed by a close parentheses and a semi-colon.

Quotation marks are NOT ALLOWED within an array element.

The URL array:

There are two ways provided for getting information from the URL, by name and by location. Which you choose to use will be defined, in part, by what the query in the URL looks like. You set this by changing the value of the variable get_by.

```
var get_by = "name"; //set as location or name.
var get_by = "location"; //set as location or name.
```

In order to use the name option, the URL must contain a name for each data element you wish to read in, and each item must be separated with an &. Make sure that the case you use matches the case in the URL.

In the name example, the query part of the URL (everything after the ?) looks like:

```
?name=<username>&password=<password>&respname=Charlie&age=45&gender=1;other=ab
```

```
{
|script type="text/javascript"|

var get_by = "name"; //set as location or name.

var urllist = new Array(
"respname",
"age",
"gender",
"other");
```

In using the location option, you can write the entire string into data and parse it later. For each item, you need to provide a starting location followed by a length, separated by a period. In determining the starting location, keep in mind that JavaScript always starts counting at zero, not one.

In the location example provided, the query part of the URL looks like:

```
?Chazz451ab&name=<username>&password=<password>
```

```
var get_by = "location"; //set as location or name.
var urllist = new Array(
"0.5",
"5.2",
"7.1",
"8.2");
```

The var array: Contains the names of the var question labels being used.

```
var varlist = new Array(
"respname",
"age",
"gender",
"other");
```

As the last part of the script, call the function with parse_query();

The full script would look like:

```
{
|script type="text/javascript"|

var get_by = "name"; //set as location or name.

var urllist = new Array(
"respname",
"age",
"gender",
"other");

var varlist = new Array(
"respname",
"age",
"gender",
"other");

parse_query();
|/script|
!disp }
```

setSubmitControl function

Purpose: To either auto submit the page after X seconds or prevent submission unless X seconds have passed after the page has loaded.

Usage: Place a call to the function in a display question at the bottom of the grid, passing the correct parameters. You can use this function only once on the same page.

```
<script type="text/javascript">
setSubmitControl(toggle,time);
</script>
```

The parameters to pass are:

toggle: the parameter can only have two values: “start” or “stop”

If “start”, the page is submitted after “time” seconds since the page loaded.

If “stop”, the page can only be submitted if “time” seconds have passed since the page loaded. You can also control the alert message presented on screen when submission is stopped, changing the message sc_msg in user_settings81.js:

```
var sc_msg = "You can submit this page only after _SECONDS_ seconds have passed.";
```

time: the time interval in seconds.

All arguments are required.

```
<script type="text/javascript">
setSubmitControl("start",10);
</script>
<script type=“text/javascript”>
setSubmitControl("stop",10);
</script>
```

The first call above will submit the page after 10 seconds. The second one will prevent submission unless 10 seconds have passed.

ShowMessageOnClose function

Purpose: To present a message to a user or interviewer when they try close or “X” out of a webSurvent survey. The default behavior for webSurvent is to not show this message. The default behavior for webCATI and the Web-based utilities is to show this message.

Usage: Change the value of the variable WS_ShowMessageOnClose to true if you want this message to be displayed in webSurvent and to false if you do not. Change the value of the variable WC_ShowMessageOnClose to true if you want this message to be displayed in webCATI or the web based utilities and to false if you do not. Additionally, if you wanted this behavior for a given page rather than for an entire survey you can set that variable to true for that page only. You could also place the same code in pagetop or pagebtm.tmpl if you wanted to make this happen for an entire survey but did not want to use a custom version of user_settings81.js

In user_settings81.js

```
var WS_ShowMessageOnClose = false;

var WC_ShowMessageOnClose = true;
```

In a page or tmpl file.

```
<script type="text/javascript">

var WS_ShowMessageOnClose = true;

</script>

<script type="text/javascript">

var WC_ShowMessageOnClose = false;

</script>
```

You can control the text of the message by changing the following variables in user_settings81.js.

For webSurvent, the message is …

var WSCloseMessage="Please do NOT close the window, answers on this page will not be saved. Instead, choose the 'Suspend' button to quit the survey";

For webCATI the message is …

var WCCloseMessage="Interviewer: Do NOT close this window or refresh the page, answers on this page will not be saved and your station will be locked until cleared. Instead, choose the 'Suspend' or 'Terminate' button to quit the survey, or choose 'Next' to continue to the end of the survey. Now, click CANCEL to cancel closing the window.";

Be careful editing these messages. They need to be on a single line and no double quotes are allowed inside the message.

NOTE: CfMC has no control over the format and layout of the box that comes up when this happens. All that can be done is insert additional text into the message.

Additionally, since there is no way to detect when a user refreshes a page or uses the browser to go forward and backward, this message will be displayed then as well. Since the software has no control over this, it is recommended that if you are not running in a pop-up window that you turn this feature off.

setForceSuspend function

Purpose: To programmatically click the suspend button after X amount of time with no activity on a page. If a user clicks any input on the page then this time is always reset back to the start. Using this function allows any data entered to be saved, unlike an auto-suspend from the CfMC system or a page time out, causing the respondents browser to go to the suspend.tmpl page.

Usage: Place this in any given page to control that page or in pagebtm.tmpl to control an entire survey. Set this slightly less than the suspend_timeout set on the CfMC system (either in the parmfile or by using the server:suspend_timeout command in super).

The syntax for the command is:

```
<script type="text/javascript">

setForceSuspend(seconds);

</script>
```

Where seconds is the number of seconds to wait before closing the page. It can be placed either in a display question at the bottom of a grid or in pagebtm.tmpl.

You can also place the following code in your suspend.tmpl to cause it to remain open instead of the default of closing after a certain amount of time.

```
<script type="text/javascript">

close_window = false;

</script>
```

NOTE: Using this approach will save any data entered on a page. The only drawback is that there is no way to differentiate between a suspend created this way and a respondent actually clicking the suspend button.

Cfmc_tmpl81.js

This new script contains the template functions as well as the no_rightclick and statbar functions.

The script must be called after cfmc_ws81.js and before user_settings81.js, either in pagetop.tmpl or header.tmpl:

```
<script src="/cfmcweb/js/cfmc_ws81.js" type="text/javascript"></script>

<script src="/cfmcweb/js/cfmc_tmpl81.js" type="text/javascript"></script>

<script src="/cfmcweb/js/user_settings81.js" type="text/javascript"></script>
```

Suspend functions

By default, a prompt is displayed when an interviewer clicks the suspend button. This prompt can be turned off and the text used can be changed by making changes to two variables in user_settings81.js

This variable controls whether the prompt is shown or not. If set to true, then a prompt will be shown. If set to false, no prompt will be shown.

```
var suspend_prompt = true;
```

You can edit the text of the message by changing the value of the following variable.

```
var suspend_msg = "Are you sure you want to suspend?";
```

Suspend template functions

Purpose: This serves multiple functions in suspend.tmpl. It writes the username of the respondent to screen (if not using the generic “Respondent” username). It dynamically determines the URL for the study and displays that to the screen. It closes the screen after a defined number of seconds when using a pop-up window. It allows you to route the respondent to a new URL after suspending. It automatically creates a link that will take a respondent back into the survey if they suspend by mistake.

Usage: This function runs automatically. There are certain elements required in the suspend template in order for the script to work properly.

You need a form named “suspend_form” with a hidden input named “suspend.” These are used by the script to determine when the suspend template is being displayed.

```
<form name="suspend_form">

<input type="hidden" name="suspend" value="true">

</form>
```

A div tag with the id="showname". This is where the Username will be filled in.

```
<div id="showname"></div>
```

A div tag with the id="showurl". This is where the URL will be filled in.

```
<div id="showurl"></div>
```

An input tag with an onClick=“restart();” This allows for re-entry into the survey.

Suspend.tmpl uses the cfmclogin form in order to resume when the user clicks on the “resume” button. The information is passed to the form from the study start html page via the cgi:

```
<form name="cfmclogin" method="post"  action="/cgi-bin/websrv77.cgi">

<!-- begin cfmc control fields -->

<input type="hidden" name="studycode"  value="@studycode">
<input type="hidden" name="use_passwds" value="yes">
<input type="hidden" name="study_dir" value="@study_dir">
<input type="hidden" name="tmpls_dir" value="@tmpls_dir">
<input type="hidden" name="cfmc"    value="@cfmc">
<input type="hidden" name="cfmccfg" value="@cfmccfg">
<input type="hidden" name="mailto"  value="@mailto">

<!-- end cfmc control fields -->

<input type="hidden" name="name" value="@name~">
<input type="hidden" name="password" value="@password~">
```

Customizing options:

There are three customizable settings in user_settings81.js:

This sets the amount of time in seconds to close the popup.

```
var suspend_secs = 30;
```

This sets the location to go to after suspending. This must be a fully functional URL. If left blank, the browser will remain on the current window.

```
var suspend_gothere;
```

To pass a respondent to another site or page you would set it as:

```
var suspend_gothere = "http://cfmc.com";
```

If you wish to pass information from the survey, then leave var suspend_gothere undefined and place the following in suspend.tmpl:

```
<script type="text/javascript">

var suspend_gothere = "http://someurl/index.html?password=@password&@somevar";

</script>
```

The password is automatically picked up through the cgi. Somevar is an HTML define created inside the spec. You can pass as many variables as you wish, although there is a 2000-character limit for the length of the URL.

Complete template function

Purpose: To close the pop-up window after a specified amount of time and to send the respondent to a different URL after completing.

Usage: This function runs automatically. There is one required page element for the function to run.

A form named "complete_form" and a hidden input named "comp":

```
<form name="complete_form">

<input type="hidden" name="comp" value="true">

</form>
```

There are two customizable settings in user_settings81.js:

This sets the amount of time in seconds to close the pop-up.

```
var comp_secs = 30;
```

This sets the location to go to after completing. This must be a fully functional URL. If left blank, the browser will remain on the current window.

```
var comp_gothere;
```

To pass a respondent to another site or page you would set it as:

var comp_ gothere = "http://cfmc.com";

If you wish to pass information from the survey, then leave var comp_gothere undefined and place the following in suspend.tmpl:

```
<script type="text/javascript">

var comp_gothere = " http://someurl/index.html?password=@password&@somevar";

</script>
```

The password is automatically picked up through the cgi. Somevar is an HTML define created inside the spec. You can pass as many variables as you wish, although there is a 2000-character limit for the length of the URL.

Terminate functions

By default, a prompt is displayed when an interviewer clicks the terminate button. This prompt can be turned off and the text used can be changed by making changes to two variables in user_settings81.js

This variable controls whether the prompt is shown or not. If set to true, then a prompt will be shown. If set to false, no prompt will be shown.

```
var terminate_prompt = true;
```

You can edit the text of the message by changing the value of the following variable.

```
var terminate_msg = "Are you sure you want to terminate?";
```

Terminate template function

Purpose: To close the pop-up window after a specified amount of time and to send the respondent to a different URL after terminating.

Usage: This function runs automatically. There is one required page element for the function to run.

A form named "terminate_form" and an hidden input named "term"

```
<form name="terminate_form">
<input type="hidden" name="term" value="true">
</form>
```

There two customizable settings in user_settings81.js:

This sets the amount of time in seconds to close the pop-up.

```
var term_secs = 30;
```

This sets the location to go to after terminating. This must be a fully functional URL. If left blank, the browser will remain on the current window.

```
var term_gothere;
```

To pass a respondent to another site or page you would set it as …

var term_gothere = "http://cfmc.com";

If you wish to pass information from the survey then leave var suspend_gothere undefined and place the following in suspend.tmpl

```
<script type="text/javascript">

var term_gothere = "http://someurl/index.html?password=@password&@somevar";

</script>
```

The password is automatically picked up through the cgi. Somevar is an HTML define created inside the spec. You can pass as many variables as you wish, although there is a 2000-character limit for the length of the URL.

The no_right click function

Purpose: To prevent right-clicking the mouse while in a survey. This stops respondents from accessing menu options that would otherwise be unavailable when running in a pop-up window. It also prevents the use of browser forward and browser backward by always loading the last page in the browser's history. Accessing the survey with a test password (first 4 chars = "test") will always allow right-clicking.

Usage: This script runs automatically. You can change the message displayed when a respondent right-clicks by changing the variable:

```
var nr_message = "This function has been disabled while in the survey";
```

in user_settings81.js you can always allow right-clicking independently of the password value by changing the variable:

```
var allow_rightclick = false;
```

in user_settings81.js; the default value for the variable is false, which allows right-clicking only with test passwords. If allow_rightclick is set to true, right-clicking is always allowed.

The pop-up help function

Purpose: To pop up an HTML page with survey instructions.

Usage: By default, the script will try to load a file called help.html from the WebArea study directory. You can change this by setting the variable var help_goto; in user_settings81.js. Please make sure that any URL path that you provide works correctly.

The statbar function

Purpose: When combined with {!statusbar qq} or {!statusbar ##}, it will create an image-driven status bar onscreen, showing the respondent a graphic representation of how much of the survey has been completed and how much remains to be done.

Usage: Generally, the statusbar is placed in either bannrtop.tmpl or bannrbtm.tmpl, depending on whether you want it displayed at the top or the bottom of the screen.

In order for the script to work, there are two items that MUST be placed in the tmpl you are using.

You must have a div tag with the id and name = "statbar". You may add any additional style elements or classes that you choose.

```
<div id="statbar" name="statbar"></div>
```

You must have a statbar form and input in order to pick up the percentage value passed by webSurvent.

```
<form name="statbar">

<input type="hidden" name="statbar_perc" value="@statusbar~">

</form>
```

Customizing the status bar

You can change the following settings in user_settings81.js to customize the look and text of the statusbar created.

```
var use_statusbar = true;   (true or false: true will display a statusbar)
var use_statusticker   = true; (true or false: true will display a statusticker)
        var statusbar_text = "<FONT SIZE=-1 color=black><i> Progress </i></FONT>";
        var statusbar_done_image = "/images/Pixels/ff9966.gif";
        var statusbar_left_image = "/images/Pixels/cc6666.gif";
        var statusbar_width = 150;
        var statusbar_align = "right";
        var statusbar_thickness = 10;
```

Here are the descriptions of the variables above:

use_statusbar – Set to true. It will not display if no value has been passed.

use_statusticker – Shows percentage done and completed in the “status”area at the bottom of the browser window.

statusbar_text – The text to be displayed with the statusbar images.

statusbar_done_image – The image to be used to show percentage completed.

statusbar_left_image – The image to be used to show percentage remaining.

statusbar_width – The width of the statusbar in pixels.

statusbar_align – The alignment to be used in the element containing the statusbar. This can only be located on the left, right or center of the screen.

statusbar_thickness – The height of the statusbar in pixels.

Adding custom scripts

You can now use your own scripts in conjunction with cfmc_ws81.js and cfmc_tmpl81.js .

In order to add your custom script, just place a call on pagetop.tmpl/header.tmpl after user_settings81.js, like so:

```
<script type="text/javascript" src="/ cfmcwe/js/cfmc_tmpl81.js"></script>
<script type=" text/javascript” src="/ cfmcweb/js /cfmc_ws81.js"></script>
<script type="text/javascript" src="/ cfmcweb/js /user_settings81.js"></script>
<script type="text/javascript” src="/ cfmcweb/js /custom.js"></script>
<script type="text/javascript">
```

There are a few rules you have to follow in order to integrate your script to work with the cfmc ones.

1. If on screen data processing is required:

a. Set a check property on the object for the function you want to call. For example, you might have:

```
your_object.check_num= true
```

2. In the same function, have the event call Check_All(this). For example, you might have something like:

```
your_object .onclick = function () { Check_All(this);}
```

3. Set up your actual check function, check_num for example.

4. Add the check function to the the main script using the AddCheckAll function in your setup function.

There are two things you can pass: the name of the function and the position of where you want it in the array: front or back. The latter is optional: if either a false or no value is passed, that function will be put at the end of the array; pass a true and it will be the first thing in the array. This is so that you can control in what order the checks occur. The values true and false must be specified as Boolean values, not strings (see example at point iii below).

So, if you had more than one function to call in your script, you should follow this order for the placement of the functions in check_all_array:

a) function that checks whether or not you can answer the question

b) function that checks if it is a valid answer

c) any other functions

For example, if check_seq below was a function that checks a certain sequence on screen and check_num was a function that checks the input validity, then you would need to call AddCheckAll for the two functions this way:

```
AddCheckAll(check_num); //in the "setnum" function
AddCheckAll(check_seq,true);//in the "setseq"
```

5. At the top of the checking function (check_num) you will need

```
if (your_object.check_num != true) return;
```

This will prevent the function from running if the object in hand doesn't need to be checked.

6. If an onload action is required, add the function calling AddOnload. For example, if you onload function was called num_onload(), then in the set up function you will need:

```
AddOnload(num_onload);
```

7. If an onsubmit function is required, add the function calling AddOnsubmit. For example, if you onsubmit function was called num_end(), then in the set up function you will need:

```
AddOnsubmit(num_end);
```

Your submit function must return true in order for the page to be submitted, false to stop submission.

You should run the custom script and call your functions only on the pages you want to. In order to do this, you can use !html_define. For example, in your qpx, you can have:

```
{!html_define custom}
{!grid}
{q1:
q text
!num,,,0-100}
{
<script type="text/javascript">
setcustom("q1");
</script>
!disp}
{!endgrid}
{!-html_define custom}
```

and ,in your pagetop.tmpl/header.tmpl you could have:

```
<script type="text/javascript" src="/ cfmcweb/js /cfmc_tmpl81.js"></script>
<script type="text/javascript" src="/ cfmcweb/js /cfmc_ws81.js"></script>
<script type="text/javascript" src="/ cfmcweb/js /user_settings81.js"></script>
>ifdef @custom
<script type="text/javascript" src="/javascript/custom.js"></script>
>endif
```

Img_81.js

This script combines various functions for manipulating images in a webSurvent survey. It provides several functions for manipulating images either on the same page or through a pop-up window. These can be used as examples of how to manipulate images using JavaScript.

The existing functions in the script are:

> changeimg - provides a simple way to change the src of an image. This change can be based on either an action taken by the user or automatically.
>
> setimg1 - provides a way to present an image on the page for a specified amount of time.
>
> setimg2 - like setimg1, but it allows you to present a pair of images rather than a single image
>
> popimg - provides a way to present images in a pop-up window. This might be used to expand a thumbnail image. The image can either be displayed for a certain amount of time or closed by the respondent
>
> showseries - provides a way to present a series of images either sequentially or randomly. The images can be shown directly to screen or in a separate pop-up window.

The changeimg function

Purpose: To change the src of a given image. This could be used to check if respondents have JavaScript or to conditionally present different images while only making one image tag.

Usage: This function can be invoked by either creating an onclick or onmouseover event on the tag itself or by calling the function separately so the change occurs automatically.

The function requires that two parameters or arguments are passed.

```
changeimg("image name","file name");
```

They are the name of the image tag in your spec, and the path/filename of the new image to be used. Each should be in quotes and separated by a comma.

In order to invoke the script automatically, first set up an image tag and then invoke the function.

```
<img name="theimage" src="/imgv1_0/sg_a.gif">

<script type="text/javascript" src="/cfmcweb/js /img81.js"></script>

<script type="text/javascript">

changeimg("theimage","/imgv1_0/sg_b.gif");

</script>
```

In order to invoke the script when a respondent interacts with the image, set the image tag up with the function call as part of the tag. In this case, the event would be clicking on the image.

```
<script type="text/javascript" src="/cfmcweb/js/img81.js"></script>

<img name="image2" src="/imgv1_0/sg_a.gif"

onclick="changeimg('image2','/imgv1_0/sg_b.gif')">
```

Please note that in this case, the arguments are surrounded by single quotes rather than double quotes.

The setimg1 function

Purpose: To present an image onscreen for a defined number of seconds. This could be used to allow a respondent to see an image for a set period of time and then ask questions about it.

Usage: Start with a cover image shown on screen. When this image is clicked, the image you wish to display is shown for the N seconds and then the original image is shown again.

This function has two parts.

The first is an onclick event handler that is placed on the image tag.

The second is invoked by a call from the JavaScript.

Call the primary script.

```
<script type="text/javascript" src="/cfmcweb/js/img81.js"></script>
```

Set up the image tag with an onclick event handler calling the subfunction changeit.

```
<img name="image1" src="/imgv1_0/sg_a.gif" onclick="changeit(this);">
```

Create another JavaScript that calls the function setimg1. This call is what sets up the relationships used by the changeit subfunction. Four arguments must be passed. Again, each argument should be in quotes and separated by commas.

```
<script type="text/javascript">
setimg1("image name","original image file","new image file","number of seconds to display");
</script>
```

The arguments, in order, are,

The name of the image tag.

The original image src file

The new image src file

The number of seconds to display the image.

```
<script type="text/javascript">
setimg1("image1","/imgv1_0/sg_a.gif","/imgv1_0/sg_b.gif","3");
</script>
```

The setimg2 function

Purpose: To present a pair of images onscreen for a defined number of seconds. This could be used to allow a respondent to see the images for a set period of time and then ask questions about them.

Usage: Start with two cover images shown onscreen. When either of the images is clicked, the images that should be displayed are shown for the N seconds and then the original images are shown again.

This function has two parts.

The first is an onclick event handler that is placed on the image tags.

The second is invoked by a call from a JavaScript.

Call the primary script.

```
<script type="text/javascript" src="/cfmcweb/js/img81.js"></script>
```

Set up the image tag with an onclick event handler calling the subfunction changeit2.

```
<img name="image1" src="/imgv1_0/sg_a.gif" onclick="changeit2(this);">
<img name="image2" src="/imgv1_0/sg_b.gif" onclick="changeit2(this);">
```

Create another script tag that calls the function setimg2. This call is what sets up the relationships used by the changeit2 subfunction. Seven arguments should be displayed (again, each argument should be in quotes and separated by commas).

setimg2("first image name","original file for 1st image","new file for 1st image","second image name","original file for 2nd image","new file for 2nd image","# of seconds to display");

The arguments, in order, are:

The name of the first image tag.

The original image src file for the first image

The new image src file for the first image

The name of the second image tag.

The original image src file for the second image

The new image src file for the second image

The number of seconds to display the image.

```
<script type="text/javascript">

setimg2("image1","/imgv1_0/sg_a.gif","/imgv1_0/sg_c.gif","image2","/imgv1_0/sg_b.gif","/imgv1_0/sg_d.gif","5");
</script>
```

The popimg function

Purpose: To present an image in a popup window.

Usage: This could be used to present thumbnail images and have a respondent click to see a larger version of the image or it could be used to popup an image automatically when the page loads. In either case you can supply a number of seconds to keep the window open or leave it open until the respondent chooses to close it. You will also need to supply parameters on how the window will be opened.

For use with an onclick function:

```
<image name="image1" src="/imgv1_0/sg_a.gif"
onclick="popimg('/imgv1_0/sg_a.gif','150','150','200','100');">
```

In the example above, no timer instructions were passed, so the window will remain open until it is closed by the respondent. You need to pass either five or six arguments, depending on whether you wish to use a timer or not. Again, each argument should be in quotes and separated by commas.

popimg('image file name','height','width','# of pixels from top','# of pixels from left','# of seconds to display');

The arguments, in order, are:

The name of the image src file to use.

The height (in pixels) of the popup window

The width (in pixels) of the popup window

The number if pixels from the top of the screen to place the popup window

The number if pixels from the left of the screen to place the popup window

The number of seconds to display the image. (*Optional*)

For use as an automatic popup.

The function call is exactly the same. Just place it at the bottom of the page (before the !endgrid). The arguments are exactly the same.

```
<script type="text/javascriptv>

popimg("/imgv1_0/sg_a.gif","150","150","600","600","5");

</script>
```

The showseries function

Purpose: To present a series of images, either on screen or in a popup window.

Usage: These images can be presented in a specific order or in random order. Each image will be displayed for a specified number of seconds.

If showing images onscreen, create a single image tag:

```
<img name="image5" src="/imgv1_0/sg_a.gif">
```

Place a script with the function call into a display question before the endgrid.

You need to pass five arguments (each argument should be in quotes and separated by commas).

```
showseries("image name","list of image files to display","use popup","randomize","# of
seconds to display");
```

The arguments, in order, are:

The name of the image tag file to use.

The list of image files to display. Each name must be separated with a colon.

Whether to use a popup window - false for no popup, true for a popup. If true, you need to present the dimensions and placement parameters for the popup window separated with a colon.

The height (in pixels) of the popup window

The width (in pixels) of the popup window

The number if pixels from the top of the screen to place the popup window

The number if pixels from the left of the screen to place the popup window

Whether to randomize the presentation: True randomizes/false does not. The number of seconds each image will be displayed.

```
<script type="text/javascript">
showseries("image5","/imgv1_0/sg_a.gif:/imgv1_0/sg_b.gif:/imgv1_0/sg_c.gif:/imgv1_0/sg_d.gif",
"false","true","5");
</script>
```

or

```
<script type="text/javascript">
showseries("image5","/giant81/sg_a.gif:/giant81/sg_b.gif:/giant81/sg_c.gif:/giant81/sg_d.gif","true:
200:200:200:400","false","5");
</script>
```

slider.js

Purpose: To input data using a slider.

Usage: Place a call to the function in a display question at the bottom of the grid. There can more than one call to the function on the page. The slider position is directly proportional to the input value. The input will be stored in either a !var or a !var,h. You also need to reference the script slider.js with a scrip tag at the top of your pagetop.tmpl or header.tmpl: <script src="/cfmcweb/js/slider.js" type="text/javascript"></script>

setslide(slider_id,range,num_decs,input_name,base_coordinates,force,input_border_style)

The parameters to pass are:

slider_id: is the id of the div element that defines the actual slider.

range: defines the min and max values we can input moving the slider. 0 <= range <= 100

num_decs: defines how many decimals you want to store in the !var. It can be 0,1 or 2.

input_name: is the name of the !var that will store the slider's value. This can be a !var,h, depending on the application. If you use a !var, the produced text boxis not clickable nd appears only when there is data stored in it.

base_coordinates: defines the slider base's – or track - width.

force: If to true forces a value onsubmit, otherwise if set to false.

input_border_style: if using a !var for the input, it allows defining the style of the border for the input box.

```
<script type="text/javascript">
setslide("slider0","0:10",0,"Q69_INPUT0","50:339","true","dashed");
setslide("slider1","1:10",1,"Q69_INPUT1","150:410","true","dashed");
setslide("slider2","1:5",2,"Q69_INPUT2","50:250","true","dashed");
</script>
```

The first call would input values from 1 to 10 with one decimal into a var question called Q69_INPUT0 and force input; the slider starts at 50px and is 310px wide.

The second call would input values from 1 to 10 with one decimal into a var question called Q69_INPUT1 and force input; the slider starts at 50px and is 310px wide.

The third call would input values from 1 to 5 with two decimals into a var question called Q69_INPUT2 and force input; the slider starts at 50px and is 310px wide.

Notes on slider elements positioning:

The elements are defined simply using CSS properties on div HTML elements. It is recommended not to use images within the div elements, but just to place a &nbsp; inside.

The initial positioning of the slider must be set with an inline style on the div, whereas other properties' values can be set in a style block in the qpx or externally in a .css file.

The two elements must be positioned with the “position:absolute” property which defines the elements positions using an absolute value.

The base must have the left and width properties defined. The slider must have the width property defined, which must be an even number. Below is an example that shows how to setup a slider.. This was taken from our on-line demo.

```
{
<table>
<tr ><td colspan="5" style="font-size: 12px;">
 This is a Fine-Grained Likert Scale, two decimals.
  <br><br>
  </td>
 </tr>
 <tr>
    <td><div style="position:absolute;left:50px;"  class="num2" >1 </div></td>
    <td><div style="position:absolute;left:100px;" class="num2" >2 </div></td>
    <td><div style="position:absolute;left:150px;" class="num2" >3 </div></td>
    <td><div style="position:absolute;left:200px;" class="num2" >4 </div></td>
    <td><div style="position:absolute;left:250px;" class="num2" >5 </div></td>
</tr>
 <tr><td colspan="5"> &nbsp;</td></tr>
<tr><td colspan="5">
 <div id="base2" class="base2" style="left:50px;width:200px;">&nbsp;  </div>
     <div id="slider2" class="slider2" style="width:12px" >&nbsp;</div>
!disp}
{!html_var_input_prefix= <div style="position:relative;left:500px;">}
{!html_var_input_suffix= </div></td></table>}
{Q69_input2:
!var,,10}
```

the used css is the following:

```
<style type="text/css">
.base2{
   line-height: 0px;
   background-color: #369;
   border-color: #def #9ab #9ab #def;
```

```
    border: 1px solid;
    border-style:inset;
    padding: 1px;
    position: absolute;
    height:4px;
}
.slider2 {
    line-height: 0px;
    background-color: #696;
    border-color: #9c9 #363 #363 #9c9;
    border: 1px solid;
    border-style:outset;
    padding: 0px;
    position: absolute;
    height:15px;
    line-height: 0;
    margin: 0;
}
.num2 {
    font-size:10px;
    text-align: center;
}</style>
```

dragdrop.js

Purpose: input data dragging and dropping elements into a defined area of the page (the “basket”);

Usage: Place a call to the function in a display question at the bottom of the grid. There can be more than one call to the function on the page. Drag is initiated by clicking on an element and drop when releasing the button. The elements snap back to their original positions if they are not dropped inside the basket area.. Only one element at a time can be dropped into the basket; the script handles element placement if one element is already in the basket. You also need to reference the script dragdrop.js with a scrip tag at the top of your pagetop.tmpl or header.tmpl:

```
<script src="/cfmcweb/js/dragdrop.js" type="text/javascript"></script>
setdrag(div_ids,img_values,basket_name,basket_id,force);
```

The parameters to pass are:

div_ids: the div id’s that are to be dragged

img_values: the values to be stored. Those values must be consistent with the div_ids.

basket_name: the name of the !var (or !var,h) where data are stored

basket_id: the id of the “basket” div element.

force: if set to true forces a value onsubmit, otherwise if set to false.

Notes on dragdrop elements positioning:

You can place an image or text inside the div elements that represent the objects to be dragged and the basket. The table below is an example of how you can setup a page. The moving elements are positioned using the CSS relative positioning property, so they all start at left:0px;top:0px; coordinates. You also have to define width and height. The basket does not need left and top defined as it is a static object.

```
<table width="50%" border="1" align="center"  padding="50">
<tr><td colspan="2">
<div id="boxA" class="box " style="left:0px;top:0px;width:100px;height:100px;">
Box A. This can contain an actual image or text.
</div>
</td></tr>
<tr><td>
<div id="boxB" class="box " style="left:0px;top:0px;width:100px;height:100px">
  Box B. This can contain an actual image or text.
</div></td>
<td><div id="boxbasket1" class="basket " style="height:150px;width:200px">
```

```
  Placeholder for Basket1. This can contain an actual image or text.
</div>
</td></tr>
<tr><td colspan="2">

<div id="boxC" class="box" style="left:0px;top:0px;width:100px;height:100px">
    Box C. This can contain an actual image or text.
</div>
</td></tr></table>

<style type="text/css">
.box {
  border: 10px solid #000000;
  position: relative;
}
.basket {
  border: 10px solid #000000;
}
</style>

setdrag("boxA,boxB,boxC","A,B,C","Q70_BASKET1","boxbasket1",”true”);
```

The function call above would manage drag and drop of elemets boxA,boxB and boxC into basket of id boxbasket1; The values stored into the !var Q70_BASKET1 would be A,B and C respectively and input would be forced

imageclick.js

Purpose: input data clicking on images.

Usage: Place a call to the function in a display question at the bottom of the grid. There can be more than one call to the function on the page. Every img tag must have a unique ID. When an image is clicked, its border thickens noticeably and becomes black.You also need to reference the script imageclick.js with a scrip tag at the top of your pagetop.tmpl or header.tmpl:

```
<script src="/cfmcweb/js/imageclick.js" type="text/javascript"></script>
```

```
setimage(img_ids,img_values,input_name,force)
```

The parameters to pass are:

img_ids: the id's of the img tags

img_values: the values to store in data when clicking on the images. Those values must be consistent with the img_ids.

input_name: the name of the !var,h (!var) where to store data

force: if left undefined or set to true forces a value onsubmit, otherwise if set to false.

```
<script type="text/javascript">
setimage("az,cr,il,ks","1,2,3,4","Q71A");
</script>
```

The function call above stores 1,2,3,4 into !var Q71A when images of id's az,cr,il,ks are clicked respectively.

dbr81.js

Purpose: To input data using a dbr-type structure with drop-downs.

Usage: Place a call to the function in a display question at the bottom of the grid. The script is currently limited to two levels plus the top level, which is an actual !fld question (drop-down). The other two levels are select html tags placed in !disp questions. You define the levels' values with two JavaScript arrays, which are defined in the same page and before the function calls. You also need to reference the script dbr81.js with a scrip tag at the top of your pagetop.tmpl or header.tmpl:

```
<script src="/cfmcweb/js/dbr81.js" type="text/javascript"></script>
```

```
setrel(level1,level2,level2data,force,namsg)
```

The parameters to pass are:

level1: the question label for the first level.

level2: the question label for the second level.

level2data: the name of the !var,h (or !var) where you want to store data.

Force: if set to true forces a response, if set to false does not force response.

Namsg: alert message text on submitting the page if there is no response in level2data and force is set to true.

Given the relative complexity of the setup, below follows a full example of page setup:

```
{!grid}
{!HTML_RADIOBUTTONS=1}

{Q72A:
Select a response from the dropdown.
!fld
== Select one response
01 Response 1
02 Response 2
03 Response 3
04 Response 4
05 Response 5
06 Response 6
07 Response 7
08 Response 8   }
```

```
{
<SELECT NAME="catr_LEVEL2" SIZE="1"  width="600" style="WIDTH: 600px"></select>
!disp}

{Q72B1:
!var,h,4}

{
<SELECT NAME="catr_LEVEL3" SIZE="1"  width="600" style="WIDTH: 600px"></select>
!disp}

{Q72B2:
!var,h,4}

{!HTML_RADIOBUTTONS=200}

{
<script type="text/javascript">
var list_width = "600px";
var level1_array = new Array(
"01:10:s1r1 level2",
"01:20:s1r2 level2",
"01:30:s1r3 level2",
"01:40:s1r4 level2",
"02:51:s2r1 level2",
"02:52:s2r2 level2",
"02:53:s2r3 level2",
"02:54:s2r4 level2",
"02:55:s2r5 level2",
"02:56:s2r6 level2",
"02:58:s2r7 level2",
"02:98:S2 Not sure",
"06:61:s6r1 level2",
"06:62:s6r2 level2",
"06:63:s6r3 level2",
"06:64:s6r4 level2",
"06:65:s6r5 level2",
"06:66:s6r6 level2",
"06:67:s6r7 level2",
"06:68:s6r8 level2",
"06:69:s6r9 level2",
"06:70:s6r10 level2",
"07:71:s7r1 level2",
"07:72:s7r2 level2",
"07:73:s7r3 level2",
"07:74:s7r4 level2",
"07:75:s7r5 level2",
"07:76:s7r6 level2",
"07:77:s7r7 level2",
"07:78:s7r8 level2",
"07:79:s7r9 level2",
"07:80:s7r10 level2",
"07:81:s7r11 level2",
"08:91:s8r1 level2",
"08:92:s8r2 level2",
"08:93:s8r3 level2");

var level2_array = new Array(
"10:1001:s1r1 level3",
"10:1002:s1r2 level3",
"10:1003:s1r3 level3",
"10:1004:s1r4 level3",
"10:1005:s1r5 level3",
"10:1006:s1r6 level3",
"10:1007:s1r7 level3",
"10:1008:s1r8 level3",
"10:1009:s1r9 level3",
"10:1010:s1r10 level3",
"10:1011:s1r11 level3",
"10:1012:s1r12 level3",
"10:1013:s1r13 level3",
"10:1014:s1r14 level3",
"10:1015:s1r15 level3",
"10:1016:s1r16 level3",
"20:2001:s2r1 level3",
"20:2002:s2r2 level3",
"20:2003:s2r3 level3",
"20:2004:s2r4 level3",
"20:2005:s2r5 level3",
"20:2006:s2r6 level3",
"30:3001:s2r7 level3",
"30:3002:s2r8 level3",
"30:3003:s2r9 level3",
"30:3004:s2r10 level3",
"40:4001:s3r1 level3",
"40:4002:s3r2 level3",
"40:4003:s3r3 level3",
"40:4004:s3r4 level3",
"40:4005:s3r5 level3",
"51:5101:s4r1 level3",
"51:5102:s4r2 level3",
"51:5103:s4r3 level3",
"51:5104:s4r4 level3",
"51:5105:s4r5 level3",
"51:5106:s4r6 level3",
"51:5107:s4r7 level3",
"51:5108:s4r8 level3",
"51:5109:s4r9 level3",
"51:5110:s4r10 level3",
"51:5111:s4r11 level3",
"51:5112:s4r12 level3",
"51:5113:s4r13 level3",
"51:5114:s4r14 level3",
"51:5115:s4r15 level3",
"51:5116:s4r16 level3",
"52:5201:s5r1 level3",
"52:5202:s5r2 level3",
"52:5203:s5r3 level3",
"52:5204:s5r4 level3",
"52:5205:s5r5 level3",
"52:5206:s5r6 level3",
"52:5207:s5r7 level3",
"52:5208:s5r8 level3",
"52:5209:s5r9 level3",
"52:5210:s5r10 level3",
"52:5211:s5r11 level3",
"52:5212:s5r12 level3",
"53:5301:s6r1 level3",
"53:5302:s6r2 level3",
"53:5303:s6r3 level3",
"53:5304:s6r4 level3",
"53:5305:s6r5 level3",
"53:5306:s6r6 level3",
"53:5307:s6r7 level3",
"53:5308:s6r8 level3",
"53:5309:s6r9 level3",
"53:5310:s6r10 level3",
"53:5311:s6r11 level3",
"53:5312:s6r12 level3",
"53:5313:s6r13 level3",
"53:5314:s6r14 level3",
"53:5315:s6r15 level3",
"53:5316:s6r16 level3",
"53:5317:s6r17 level3",
"53:5318:s6r18 level3",
"53:5319:s6r19 level3",
"54:5401:s7r1 level3",
"54:5402:s7r2 level3",
"54:5403:s7r3 level3",
"54:5404:s7r4 level3",
"54:5405:s7r5 level3",
"55:5501:s8r1 level3",
"55:5502:s8r2 level3",
"55:5503:s8r3 level3",
"55:5504:s8r4 level3",
"55:5505:s8r5 level3",
"55:5506:s8r6 level3",
"55:5507:s8r7 level3",
"55:5508:s8r8 level3",
"56:5601:s9r1 level3",
"56:5602:s9r2 level3",
"56:5603:s9r3 level3",
"56:5604:s9r4 level3",
"56:5605:s9r5 level3",
"56:5606:s9r6 level3",
"56:5607:s9r7 level3",
"56:5608:s9r8 level3",
"56:5609:s9r9 level3",
"56:5610:s9r10 level3",
"56:5711:s9r11 level3",
"56:5712:s9r12 level3",
"56:5713:s9r13 level3",
"56:5714:s9r14 level3",
"56:5715:s9r15 level3",
"58:5801:s10r1 level3",
"58:5802:s10r2 level3",
"58:5803:s10r3 level3",
"58:5804:s10r4 level3",
"58:5805:s10r5 level3");
setrel("Q72A","LEVEL2","Q72B1","true","Please provide a response at each level");
setrel("LEVEL2","LEVEL3","Q72B2","true","Please provide a response at each level");
</script>
!disp}
{!end_grid}
```

placefocus.js

Purpose: To place a focus on the first visible input tag on the page or on the first button if there are no visible inputs on the page.The script can be useful in webCATI if the interviewers want to input answers only using the keyboard.

Usage: Place a call to the script in pagetop.tmpl, after the standard cfmc scripts:

```
<script src="/cfmcweb/js/cfmc_ws81.js" type="text/javascript"></script>
<script src="/cfmcweb/js/cfmc_tmpl81.js" type="text/javascript"></script>
<script src="/cfmcweb/js/user_settings81.js" type="text/javascript"></script>
<script src="/cfmcweb/js/placefocus.js" type="text/javascript"></script>
```

You can toggle the call to placefocus.js using !html_define in the qpx.

The script will run only when in the survey, i.e. whenever the form “cfmc” is found.

On some older browsers, focus placement might not be supported on radio buttons and checkbox inputs; if that is the case and the first unhidden input on the page is either a radio button or a checkbox, then the script will not pace focus on any input.

Changing the physical position of a button can impact focus placement. For example, the script places focus on the “start interview” button on the default qprompt.tmpl, since the button is the first that the script finds. If the first button is the “quit interview” one, then focus will be placed on that. If you use the float CSS property, you can move buttons on the page while still keeping their physical positions fixed.

Appendix I

ADA COMPLIANCE

Overview

Accessibility standards are checklists of things that must/should be done to Web pages. These include using formatting that allows software interpreters to process the html without confusion, such as proper use of table headers and table data tags. They also include things like requiring alternate text to go along with audio and visual content.

There are two major standards for accessibility compliance. The first is WCAG (Web Content Accessibility Guidelines). It is maintained by the same group that defines the html standards. The official WCAG site looks like:

http://w3c.org/WAI

The second is the U.S. Government guidelines. If you meet these guidelines, your site is said to be Section 508 compliant. A good Web site for the Section 508 guidelines is:

http://www.access-board.gov/sec508/guide/1194.22.htm

The WCAG standard has a bit more jargon associated with it. They break their checklists into different levels, Priority 1-3. Compliance said to be A, AA, or AAA, where A means that all Priority 1 items are done, AA means that all Priority 1 and 2 items are done, and AAA means that all items are done.

All items in the Section 508 guidelines appear in the WCAG checklists, but is an odd mix of some (but not all) Priority 1 items and some Priority 2 and 3 items. That means that a page that is WCAG AAA compliant is also Section 508 compliant, but a page the is Section 508 compliant is not WCAG compliant at any level.

Validating Compliance

Instead of going through the checklists manually, the easiest way to check a page for compliance is to use a validator. There are online, such as Bobby …

http://bobby.watchfire.com/bobby/html/en/index.jsp

… where you enter the URL of your page, choose your desired compliance standard, and get a report about your page. There are other tools available, some which run on windows and some which run on UNIX.

Windows: http://www.chami.com/html-kit

UNIX: http://tidy.sourceforge.net

Many html editors/validators have accessibility checking built in.

Go to http://downloads-zdnet.com.com

CfMC is currently working to develop tools to help you use these validators to make sure your surveys are compliant.

Section 508 Checklist

A text equivalent for every non-text element shall be provided (e.g., via "alt","longdesc", or in element content).

Equivalent alternatives for any multimedia presentation shall be synchronized with the presentation.

Web pages shall be designed so that all information conveyed with color is also available without color, for example from context or markup.

Documents shall be organized so they are readable without requiring an associated style sheet.

Redundant text links shall be provided for each active region of a server-side image map.

Client-side image maps shall be provided instead of server-side image maps except where the regions cannot be defined with an available geometric shape.

Row and column headers shall be identified for data tables.

Markup shall be used to associate data cells and header cells for data tables that have two or more logical levels of row or column headers.

How can HTML tables be made readable with assistive technology?

Using the "Scope" Attribute in Tables – Using the "scope" attribute is one of the most effective ways of making HTML compliant with these requirements. It is also the simplest method to implement. The scope attribute also works with some (but not all) assistive technology in tables that use “colspan” or “rowspan” attributes in table header or data cells.

Using the Scope Attribute – The first row of each table should include column headings. Typically, these column headings are inserted in <TH> tags, although <TD> tags can also be used. These tags at the top of each column should include the following attribute:

scope="col"

By doing this simple step, the text in that cell becomes associated with every cell in that column. Unlike using other approaches (notably "id" and "headers") there is no need to include special attributes in each cell of the table. Similarly, the first column of every table should include information identifying information about each row in the table. Each of the cells in that first column are created by either <TH> or <TD> tags. Include the following attribute in these cells:

scope="row"

By simply adding this attribute, the text in that cell becomes associated with every cell in that row.

While this technique dramatically improves the usability of a Web page, using the scope attribute does not appear to interfere in any way with browsers that do not support the attribute.

Example of source code:

The following simple table summarizes the work schedule of three employees and demonstrates these principles.

```
<table>
<tr>
<th>&nbsp;</th>
<th scope="col" >Spring</th>
<th scope="col" >Summer</th>
<th scope="col" >Autumn</th>
 <th scope="col" >Winter</th>
 </tr>
<tr>
<td scope="row" >Betty</td>
<td>9-5</td>
 <td>10-6</td>
 <td>8-4</td>
<td>7-3</td>
</tr>
<tr>
 <td scope="row" >Wilma</td>
<td>10-6</td>
 <td>10-6</td>
 <td>9-5</td>
 <td>9-5</td>
</tr>
<tr>
 <td scope="row" >Fred</td>
<td>10-6</td>
 <td>10-6</td>
 <td>10-6</td>
 <td>10-6</td>
</tr>
</table>
```

This table would be displayed as follows:

	Spring	Summer	Autumn	Winter
Betty	9-5	10-6	8-4	7-3
Wilma	10-6	10-6	9-5	9-5
Fred	10-6	10-6	10-6	10-6

The efficiency of using the scope attribute becomes more apparent in much larger tables. For instance, if an agency used a table with 20 rows and 20 columns, there would be 400 data cells in the table. To make this table comply with this provision without using the scope attribute would require special coding in all 400 data cells, plus the 40 header and row cells. By contrast, using the scope attribute would only require special attributes in the 40 header and row cells.

Frames shall be titled with text that facilitates frame identification and navigation.

Pages shall be designed to avoid causing the screen to flicker with a frequency greater than 2 Hz and lower than 55 Hz.

A text-only page, with equivalent information or functionality, shall be provided to make a Web site comply with the provisions of these standards, when compliance cannot be accomplished in any other way. The content of the text-only page shall be updated whenever the primary page changes.

When pages utilize scripting languages to display content, or to create interface elements, the information provided by the script shall be identified with functional text that can be read by assistive technology.

When a Web page requires that an applet, plug-in or other application be present on the client system to interpret page content, the page must provide a link to a plug-in or applet that complies with §1194.21(a) through (l).

When electronic forms are designed to be completed on-line, the form shall allow people using assistive technology to access the information, field elements, and functionality required for completion and submission of the form, including all directions and cues.

While the relationship between the titles "First Name" or "Last Name" and their respective input boxes may be obvious from visual inspection, the relationship is not obvious to a screen reader. Instead, a screen reader may simply announce "input box" when encountering each input box. The reason for these difficulties is revealed from inspecting the HTML source for this table. The following code is a simplified version of this table.

```
<FORM>
<TABLE>
<TR>
<TD><B>FIRST NAME: </B></TD>
<TD><INPUT TYPE="TEXT" NAME="FIRSTNAME"> </TD>
</TR>
<TR>
<TD><B>LAST NAME: </B></TD>
<TD><INPUT TYPE="TEXT" NAME="LASTNAME"> </TD>
</TR>
```

```
</TABLE>
<P>
</FORM>
```

Use the <LABEL> Tag and Associated "FOR" Attribute to Tag Labels. In other words, identify the exact words that you want to use as the label for the form element and enclose those words in a <LABEL> tag. Use the "FOR" attribute to uniquely identify that element.

Use the "ID" Attribute in the Associated Form Element. Every form element supports the "ID" attribute. By setting this attribute to the identifier used in the "FOR" attribute of the associated <LABEL> tag, you "tie" that form element to its associated label. For instance, we have rewritten the HTML code for our simple form-inside-a-table to include explicit labels below. The new HTML code for the explicit labels is indicated in bold:

```
<FORM>
<TABLE>
<TR>
<TD><B><LABEL FOR="first"> FIRST NAME:</LABEL> </B></TD>
<TD><INPUT TYPE="TEXT" NAME="FIRSTNAME" ID="first" ></TD>
</TR>
<TR>
<TD><B><LABEL FOR="last"> LAST NAME:</LABEL> </B></TD>
<TD><INPUT TYPE="TEXT" NAME="LASTNAME" ID="last" ></TD>
</TR>
</TABLE>
</FORM>
```

In a nutshell, that's all there is to making HTML form elements accessible to assistive technology. Experience has shown that this technique works extremely well in much more complicated and convoluted forms and it should work well in all agency HTML forms.

A method shall be provided that permits users to skip repetitive navigation links.

When a timed response is required, the user shall be alerted and given sufficient time to indicate more time is required.

CfMC and labels

WebSurvent and webCATI automatically output a label and an ID for each question. For var and text questions, the label is placed on around the question text and the id is equal to the question label. For fld and cat type questions, the label is placed around the response text and the ID for each response is the question label and the response code. For num questions, the two methods are combined. The question text is surrounded by a label and the id is placed on the input tag for the numeric entry. Additionally, if there are exception codes a label is placed around the text of each exception code and the ID for the input is the question label and the code.

To turn off automatic generation of labels, use:

```
{-!html_labeltags}
```

This can be done for the entire spec or it can be turned off as needed.

Var/Text questions

```
<label for="EMAIL2">
Email address previously.
</label>
<INPUT TYPE="text" NAME="var_EMAIL2" id="EMAIL2" SIZE="70" maxlength="70"
VALUE="">
```

Fld/Cat questions

```
S2a. Do you think that demonstration surveys are entertaining?
 <INPUT TYPE="radio" NAME="catr_S2A" id="S2A_1" VALUE="1"><label
for="S2A_1">Yes</label>
<INPUT TYPE="radio" NAME="catr_S2A" id="S2A_2" VALUE="2"><label
for="S2A_2">No</label>
<INPUT TYPE="radio" NAME="catr_S2A" id="S2A_3" VALUE="3"><label
for="S2A_3">Don't know</label>
```

Num questions

```
<label for="S2C">
S2c. How many years have you been programming CFMC software?
</label>
<INPUT TYPE="text" NAME="num_S2C" id="S2C" VALUE="" SIZE="2" maxlength="2">
<INPUT TYPE="checkbox" NAME="rnum_S2C" id="S2C_DK" VALUE="DK">
<label for="S2C_DK">Don't Know</label>
<INPUT TYPE="checkbox" NAME="rnum_S2C" id="S2C_LT" VALUE="LT">
<label for="S2C_LT">Less than one year</label>
<label for="S2C">
```

Appendix J

INSTALLATION ISSUES

Minimum Hardware Requirements

- A computer to handle as many concurrent interviews as needed.

- Internet/Intranet connection sufficient to handle the volume expected.

- Web server software (usually Apache)

- A reasonably contemporary CPU (e.g., one that has a Pentium chip). At least 128 MB of RAM (estimate about 2.5 megabytes per webCATI session).

- 10-GB partition on hard drive devoted to CfMC applications (/cfmc). 10 gigabytes is actually more than enough. If it is not possible to make a partition, install in a partition ther than root. The reasoning behind this is if you fill up the disk using CfMC software, you'll only crash the CfMC's software, not the machine.

NOTE: DO NOT install the software on a network-mounted partition. There may be file-locking problems that cannot be avoided.

- Fast Internet Connection - T1, if possible. (3com network cards are preferred. They seem to have fewer traffic problems.)

- Telnet/ssh capabilities or a dedicated modem (for troubleshooting)

NOTE: Although Survent runs on several different operating systems, including DOS, MPE and UNIX, CfMC's Web programs only run on a UNIX operating system at this time. All the instructions in this documentation are for a UNIX operating system.

Minimum software requirements

- Linux distribution with kernel 2.0 or higher, SCO OpenServer 5.0.X, HPUX 11.0 or higher, Solaris 2.6 or higher, or AIX Operating System.

- Web server software: Apache version 1.3.2 or higher is recommended.

- Survent for UNIX, Release 8.1 or higher.

- PERL version 5 or higher.

NOTE: All reasonably contemporary UNIX installs will have this installed already. The perl program is looked for in the directory */usr/bin/* (such as, "/usr/bin/perl"). If perl exists but it is not in */usr/bin/*, set a symbolic link to wherever it exists. (for example, "ln -s /bin/perl /usr/bin/perl"). If perl does not exist on your system, you may download it for free from http://www.cpan.org/ports/, install instructions are provided.

- PHP version 4.2 or higher.

Setting up users

The preferred command interpreter is /bin/csh (c-shell). This is the only shell we can fully support you with. Some users use other shells which may be ok.

Logging in (.login):

Type:

```
umask 000
setenv CFMC <wherever the software was installed>/
(e.g., "setenv CFMC /cfmc/", the last "/" is far too important)
setenv CFMCCFG ${CFMC}ipcfiles/
setenv PATH .:/bin:/usr/bin:/usr/local/bin/:${CFMC}go/
```

The path can include anything desired, but make sure that ${CFMC}go/ is included $CFMC is used widely in the software to define where to look for particular functional files necessary to run the software.

$CFMCCFG defines where to look for necessary communication files.
The PATH allows you to run any of the programs in the ${CFMC}go/ directory from any other directory.

NOTE: See ${CFMC}control/login.exam for more detail.

User Types

User Name	Description	Group
cfmc	software owner	cfmc
server	study server	cfmc
super	supervisor	cfmc

If using telnet sessions (i.e. non-hardwired):

User Name	Description	Group
intv###	interviewers	<not cfmc>
code###	coders	cfmc

If using hardwired terminals:

User Name	Description	Group
intv	interviewers	<not cfmc>
code	coders	cfmc

Others (any user name)

Description	Group
spec writer	cfmc
unsupervised	cfmc

Note about ### for telnet interviewer and coder accounts:
Specify the station number/ldev in the signon name (e.g., intv101) and put the following in .login:

```
setenv LDEV `echo $LOGNAME | cut -c 5-7`
@ sock = 3000 + $LDEV
setenv SOCKET $sock
```

File permissions for important directories

NOTE: Directories listed are SUB-directories beneath your main cfmc install directory, unless otherwise noted.

Core directories

drwxr-xr-x	2 cfmc cfmc	4096 Apr 30 14:01	go (scripts, compiled programs (binaries))	
drwxr-xr-x	2 cfmc cfmc	4096 Apr 30 14:01	control (message file for programs, parameter files, initializations files)	
drwxr-xr-x	2 cfmc cfmc	4096 Apr 30 14:01	support (utility qffs)	

Study file directories

drwxrwxrwx	3 cfmc cfmc	4096 Apr 30 14:01	data	(data(.tr) (initially created when study is brought up) dbfiles from compile of questionnaire spec)
drwxrwxr-x	2 cfmc cfmc	4096 Apr 30 14:01	fone	(phone sample files/ phone index files created from fonebuld)
drwxrwxr-x	2 cfmc cfmc	4096 Apr 30 14:01	qff	(compiled questionnaire spec file)

```
 drwxrwxr-x     2 cfmc  cfmc    4096 Apr 30 14:01   quota   (quota files from compiled
                                                            questionnaire spec)
drwxrwxrwx     4 cfmc   cfmc    4096 Apr 30 14:01   resume  (suspend files to be resumed,
                                                            created during interview)
```

Communication directories

```
 drwxrwxrwx    2 cfmc   cfmc    4096 Apr 30 14:01   ipcfiles (local configuration files)
 drwxrwxr-x    2 cfmc   cfmc    4096 Apr 30 14:01   super     (study server log files)
 drwxrwxr-x    2 cfmc   cfmc    4096 Apr 30 14:01   /cfmc/cfg (global configuration files)
```

Reasons for CfMC group write access

data,fone,qff,quota: For loading/creating/updating study files

ipcfiles,resume,super: For troubleshooting

Reasons for open write access

data: Interviewer has to be able to make blow (.b_) directory if necessary

resume: Interviewer has to be able to make resume (.r_) directory if necessary

ipcfiles: Interviewer has to be able to create ipc file (wipc####.pub) and log file (log<intvid>) if necessary.

File permissions for important files

<study>.qff: rw-rw-r--

<study>.quo: rw-rw-r--

<study>.fon/fnx: rw-rw-r--

<study>.tr: rw-rw-r--

Files in <study>.r_: rw-rw-r--

Installing the software

Sign on to your Linux machine as the user that should own the software. This user should sign on to the c-shell (/bin/csh). DO NOT SIGN ON AS “ROOT.” See “Setting up users” above for important information before continuing.

If you are installing over a precious version, make sure that you have a backup of your present CfMC world and make sure there are no CfMC programs running (type “ps auxw | grep $CFMC”). If you have the Survent or webSurvent package, kill any processes such as snap81, stdysrvr, survsupr, survmon and survent (“kill -9 <processid>”).

Make and/or go to the directory where you want the software to exist (e.g. ““mkdir /usr/cfmc” then “cd /usr/cfmc”)

> **NOTE 1**: It is recommended that the software be installed on a partition other than the / (or root) partition. 10 gigabytes of free space should be enough.
>
> **NOTE 2:** Even if you're not installing the software in the /cfmc directory, make sure the /cfmc directory exists and is writable to by the cfmc owner (from Step 1); the /cfmc directory will be used. Don't make this directory a symbolic link to your main CfMC software directory. The software will not function properly.
>
> **NOTE 3:** The Web server’s http daemon will need to have read access to certain files in your CfMC software area, so, make sure that the directory defined above is readable by it. It is simplest to make the directory world-readable: by typing:

```
chmod –R 755   <directoryname>
```

Place the qzipped tar software file (e.g., linux.ws.81_####.tar.gz) in the directory created in step 3.

Uncompress the file (e.g. gzip-d linux.ws.81_####.tar.gz).

Untar the file (e.g. tar xvpPf linux.ws.81_####.tar).

Run the cleango command file to perform some minor installation tasts (e.g. “./cleango”).

Edit/create /cfmc/cfg/validate and add your expiration and validation strings (if applicable).

9. Run wssetup to complete the webSurvent install (e.g., "./wssetup").

   NOTE FOR WebCATI/WebSuper CLIENTS: Use the "wc" option (e.g., "./wssetup wc").

   NOTE FOR WPM CLIENTS: Use the "wpm" option (e.g., "./wssetup wpm"). You will have to be able to write to your main Web Server document area for this so you may have to sign on as root to perform this. Be sure you know where your main html Web Server (httpd) document and cgi-bin areas are BEFORE you run wssetup.

If you have any problems or questions, please email us (support@cfmc.com) or give us a call (415-777-2922).

NOTE: If your previous version was 7.7 compile #2676 or earlier, several important changes have been made:

- websrv81.cgi is installed and looked for in /cgi-bin/cfmccgi/ instead of /cgi-bin/

- javascript, images, css and php files are stored in subdirectories of /cfmcweb/ in the html document area If you want to run any old studies with this new software, be sure to update your index.html's accordingly

Configure webCATI to use socket (tcp/ip) communications between the server and interviewing processes by adding the following entry to:

```
${CFMC}control/parmfile:sockets: yes
```

This sets up communications between Survent and the CfMC server, using TCP/IP protocol (sockets).

It also copies JavaScripts for automatic input evaluation to your Web server area. (The JavaScript and images Web server directories should have read access for all users.)

WebCATI installation checking notes

It is important that you do these steps in the order specified here. Many of the steps outlined here will not work if the previous steps have not been successful. If any step fails, be sure to determine why it is failing and fix it before going on. Also, it is assumed that ***http://distrib.cfmc.com/support/hints/setup.html*** has been read and understood. Since this area is password protected and the password changes periodically, you will have to contact support@cfmc.com to get the password.

Test the webserver httpd process:

Check to see that the httpd daemon is running by typing:

```
% ps auxw | grep httpd (For Linux)
% ps –ef | grep httpd (For other types of UNIX)
```

If httpd is not running, start it (Consult your System Administrator if necessary).

Installation confirmation

Confirm that the $PATH $CFMC and $CFMCCFG environment variables have been set appropriately in the .login script. Type:

```
echo $PATH
echo $CFMC
echo $CFMCCFG
```

The PATH should have <cfmcloc>go/ defined in it first; e.g. setenv PATH ${CFMC}go:$PATH

CFMC should be the <cfmcloc>

CFMCCFG should be <cfmcloc>/ipcfiles/

Confirm that the directory that they software was installed in is owned by one user, e.g., “cfmc”. Also confirm that this directory and super directories (e.g. the super directory of /usr/cfmc is /usr) have the necessary read/write/execute status (rwxr-xr-x). To set this for the cfmc directory, enter “chmod –R 755 ${CFMC}/*”.

Also confirm that /cfmc and /cfmc/cfg are owned and writable to by this same user. Use chown -R <user> <directory> to recursively change the ownership of these directories, their subdirectories and the files, and “chmod –R 755 /cfmc” to set the permissions. You must be logged on as the owner or root to do this.

Confirm that websrvxx.cgi exists in the Web server’s cgi-bin directory and it can be executed by all. (rwxr-xr-x access). The cgi-bin directory can be found in the httpd configuration file (/etc/httpd/conf/httpd.conf) as “ScriptAlias /cgi-bin/”. It is usually in the same directory as the default directory for Web server documents (where the default index page is when a browser goes to the site).

Confirm that the ${CFMC}control/parmfile contains the following line:

```
sockets: yes
```

This line should have been added to the parmfile when wssetup was run during the webCATI software installation.

Running the software

Try running the Mentor program.Type “mentor con con”. If “command not found” comes up, the PATH is wrong. If a “no message file” error comes up, the CFMC variable is wrong. If things run fine, you should get the Mentor header and be prompted for a tilde (~) command. Type “~end” to exit Mentor.

Try running the study server. Type “server bg”. The process ID for the study server should be returned to the screen and the UNIX prompt should be returned as well. If not, check the file ${CFMC}super/log.serv and ${CFMC}super/log.snap for some clues. If this doesn't help, try running the server again to the screen by typing “server 901”. If you discover that the snapper process isn't running, try running the snapper to the screen (type “snap81 log”). Generally, a problem like this is due to read/write access problems. The files to check are ${CFMC}super/: log.snap, cfmcsnap.lst, ${CFMC}ipcfiles/*.cfg, /cfmc/cfg/*.cfg, /cfmc/cfg/dumplog.

Try running the supervisor program. Type “super 234”. You should see the main Super menu. If not, perform the debugging techniques described in the previous step. If successful, type <ctrl>-Break and “quit” to quit out. If you are stuck in the Supervisor and cannot type a command, your

“Break” character is a different character or undefined. Look in the .login script to see what it says on the line “stty intr x”, “x” should be the break key.

Bringing up interactive test Survent job

To make sure the study server and application are running independent of the webserver, we will make a test study to run. Make a study directory and place yourself in it. (e.g., “mkdir rrunr” then “cd rrunr”).

Copy the specfile “rrunr.qpx” from the CfMC Survent examples directory (e.g., “cp ${CFMC}survent rrunr.qpx .”).

Compile the questionnaire. Type "mentor rrunr.qpx -rrunr.lfl". Errors would be presented to the screen but there should not be any.

Load the study files. Type “loadstudy rrunr testing”. Check for errors; generally these will have to do with read/write access.

Create the file employee.xxx in the ${CFMC}ipcfiles/ directory. There is an example file emplexam.xxx that you can use, or you can create one of your own. This will hold the interviewer ids that you would use when testing studies. If you use the provided emplexam.xxx file, you can use the ID “DBUG” to do most testing functions.

Try to start an interview. Type “netsurv 234” (the “234” is the ldev number, this can be any number from 2-2000, just don't use 901 because that's the ldev number your study server is using). If netsurv cannot be found, check your $PATH environment variable as noted in step 2. Once pressing return after the LDEV number confirmation, you should be prompted for a qff file name. If you are not, check your $CFMC environment variable as noted in step 2.

Enter “rrunr” at the "Type questionnaire (QFF) file name →" prompt

Enter “dbug” or the ID of your choice when prompted for an interviewer ID. If the employee.xxx file cannot be found, break out of the session and check the definition of your $CFMCCFG variable (type "echo $CFMCCFG"). See step 2 for proper environment variable definitions.

The study server should be detected at this point and you should get the "(RRUNR) Press to interview, or Q)uit→" prompt. If you do not, check the $CFMCCFG variable definition as noted in step 2. Make sure that this user has write privileges to the $CFMCCFG directory.

As instructed, Press <Enter> to start the interview.

Make sure an interview can be suspended and resumed correctly. Type “suspend” at some question, then enter the name of the file to save, then type “Resume” at the next interview prompt. It's generally a read/write/execute problem with ${CFMC}resume/ if the survey can't be resumed.

Bringing up a webCATI job using a browser

Go to ${CFMC}websurv/studies/wctest.

Compile the questionnaire. Type “mentor wctest.qpx -wctest.lfl” Compile errors would be presented to the screen if there were any, but in this case there should not be any.

Create a phone file to be used for testing with the following command makefone <studycode> <ASCII sample file> <optional :fonetextlength>

Load the study files. Type “loadstudy wctest”. Check for errors, generally these will have to do with read/write access. Follow the loadstudy prompts when necessary.

Create the index.html for the job by typing “wcatutil wctest default”. This will not only create the index.html but it will place the file in a directory called wctest in the html document directory in the web directory of the machine. The Web area will be defined in /cfmc/cfg/webinfo. If it's not, check with the installer to see where it is or try to find the file httpd.conf and look for the “DocumentRoot” definition. If wcatutil cannot create a study directory, either have the system manager check the read write access of the Document root area to allow your user write access or sign on as root. Do either, then type "movehtml" in the wctest study directory.

Bring up a browser and go to the URL for the study (http://www.whatever.com/wctest or if there is no domain yet (http://<ipaddress (e.g., ###.###.###.###)>/wctest/). The last slash is important when using the ipaddress. NOTE: If you have no domain name and the ipaddress has to be used, edit index.html in the study directory and replace the <hostname> in “ACTION="http://<hostname>/cgi-bin/websrv81.cgi>” with the ip address.

Run movehtml again to reload the index.html. Be sure to refresh the page when you go back to your browser.

Try to start an interview. If the next page cannot be found, check to make sure that websrv<ver#>.cgi exists in the cgi-bin directory and it can be executed by all. The cgi-bin directory can be found in httpd.conf as “ScriptAlias/cgi-bin/ <directory>.

Additional parmfile options

You may modify the following options in the ${CFMC}control/parmfile file. The settings apply to all studies running under the server.

Setting maximum number of concurrent Web surveys (HTML limit)

HTML_LIMIT:#

This option limits the number of concurrent webSurvent sessions for the CfMC study server. (# can be zero to the maximum allowed in your contract.) Always set the HTML limit to less than the maximum stipulated in the contract and allow for regular telephone Survent interviews as well as webSurvent interviews for the same study.

For example, if the contract stipulates a maximum of 35 sessions and there are 20 interviewers conducting Survent telephone interviews, set the html_limit to 15. The 16th concurrent webSurvent session will not be allowed into the survey until one of the other webSurvent interviews is finished, to ensure that the 35-interview limit will not be exceeded.

> **NOTE:** You **MUST** set your HTML limit in your parmfile in order to let more than one person into a survey at a time. So, do not forget this important step.

SUSPEND_TIMEOUT

There are two controls for the use of SUSPEND_TIMEOUT. There is the parmfile option that is set for all studies:

```
suspend_timeout:15
```

Then there is a supervisor option that lets you change the command globally or for a specific study. If the study server is taken down, then these settings will revert to what is specified in the parmfile.

The syntax is:

```
server:suspend_timeout = ##

or

server:suspend_timeout:<study> ##
```

In version 7.7+, the timeout is based on when a respondent last communicated with the study server. (In version 7.6, it was based on when the respondent initially started the session.)

Setting background Survent processes (optional)

```
HTML_SURVENTS_BACKGROUND:#
```

This option sets the number of Survent processes to keep running in the background waiting for webSurvent users to log on and speeds up the initial loading f Survent for the webSurvent users. It is not necessary to modify this option unless you expect to be running several hundred interviews concurrently.

Setting direct CGI to Survent connection (optional)

```
CGI_DIRECT_TO_SURVENT:YES
```

This determines whether the cgi process connects directly to Survent (yes) or goes through the study server (no). Setting this option to yes increases the speed of processing by bypassing the study server, however the Survent session will not be logged in the study server's logfile (ll###### found in ${CFMC}super/). Use this option if you experience slowdowns during interviews.

Appendix Y

OPEN FILES LIMIT

NOTE: Many types of Linux exist besides RedHat, such as Debian, Mandrake, Novell/SUSE and Slackware. If you decide to use any Linux OS installation other than RedHat Version 8 or higher, please keep in mind that even though our Linux version should compile and run on it, thorough testing of CfMC's software may not have been performed on that operating system. Based on our experience with the other types of Linux, most problems occur due to system settings that must be changed. Your technical staff will know your system far better than CfMC, but CfMC will do its best to help you troubleshoot any problems.

To check how many open files your kernel is able to handle, type whichever path applies to your version of RedHat:

For most recent versions of RedHat

This will increase the limit of the number of concurrent open files **for csh (c-shell) sessions only** for RedHat System 8 and above, plus Redhat Enterprise ES version 3.

Sign on as root and enter the following:

```
# vi /etc/security/limits.conf
```

Add:

```
*   soft   nofile   32768
*   hard   nofile   32768
```

```
# vi /etc/pam.d/login
```

Add the line:

```
session required /lib/security/pam_limits.so
```

```
# vi /etc/pam.d/sshd
```

Add the line:

```
session required /lib/security/pam_limits.so
```

```
# vi /etc/ssh/sshd_config
```

Enable:

```
UsePrivilegeSeparation no
# service sshd restart
```

Earlier versions of RedHat

For older versions of Red Hat, to check how many open files your kernel is able to handle, type whichever path applies to your version of Red Hat:

```
Cat/proc/sys/f8/file-max
```

Change the following configuration files:

Add these lines to the /etc/security/limits.conf

```
* soft nofile #### (file limit)
* hard nofile #### (file limit)
```

Determine your file limit according to the Kernel version that you have. The file limit should be three times the maximum number of files indicated for the Kernel. For example, if the Kernel limit is 8192, then the file limit would be 24576.

Add this line to etc/pamd/login:

```
session required/lib/security/pam_limits.
```

Appendix Z

PROXYPASS

Using ProxyPass to allow webSurvent respondents inside your firewall

What it does

ProxyPass allows machines behind your firewall that have local network IP addresses (e.g. 192.168.x.x) to serve Web pages to the outside world. For webSurvent, this means that additional machines can be added to your site with a minimum of effort and expense. For example, additional "real" IP addresses do not need to be acquired.

This document was written with Linux and Apache in mind. However, other OS/server combinations may also be able to take advantage of the PROXYPASS webSurvent variable. The webSurvent cgi knows by what name it was called. The ProxyPass string is prepended to this name with trailing slashes are removed.

Requirements

The webSurvent ProxyPass option takes advantage of the Apache ProxyPass module. Therefore, you must be running Apache on the "outer" machine (i.e., the one with the real IP address and domain name). The outer machine must also be running IP masquerading of some variety so that machines behind the firewall can make contact with the outside world. Most Linux distributions ship with both Apache and IP masquerading. Proxypass has been tested using Linux machines. There are probably other configurations that will work also.

The other requirement is a functioning LAN with access to the Internet. A common simple setup is a Linux box with a static IP address and domain name, connected via DSL to the Internet, running IP masquerading and Apache. This machine is connected to an Ethernet hub, to which machines having local IP address are connected. Some or all of the local machines run http servers.

Getting started

Here's the express version:

Add a line resembling the following to your outer machine's httpd.conf file

```
ScriptAlias /ws1/cgi-bin/ "/home/httpd/cgi-bin/"
```

Add a line resembling the following to your inner machine's httpd.conf file.

```
ScriptAlias /ws1/cgi-bin/ "/home/httpd/cgi-bin/"
```

Edit your webSurvent .html files so that references to /cgi-bin/ have the ProxyPass name prepended, as in:

```
<FORM name="cfmclogin" METHOD="POST" ACTION="/ws1/cgibin/websrv81.cgi">
```

Add a setting for PROXYPASS wherever you set CfMC in your .html files:

```
<INPUT type="hidden" NAME="PROXYPASS" VALUE="/ws1/">
```

Getting the latest version of Apache

Apache does not ship with ProxyPass enabled, nor is it included by default when you compile Apache yourself.

Get the latest version of Apache from http://www.apache.org and follow the instructions provided for compiling it with the module named "mod_proxy" included. Note that there is an option to include all of the modules. The Apache Web site is excellent. If you follow instructions, you are unlikely to have difficulty with this step.You will be able to tell if ProxyPass works with your version of Apache simply by trying to use it (see below). Add a "ProxyPass" line to your httpd.conf file, stop and start the server. If ProxyPass is not available, you'll get an error message stating something to that effect.

Once you are running Apache with mod_proxy installed on your outer machine, add a line similar to the following to Apache's httpd.conf file on the outer machine (at the end of the file is fine).

```
ProxyPass /ws1/ HTTP://192.168.1.11/
```

In the above example, ws1 is just a name that will be used later in URLs to reference the machine whose local IP address matches the second argument. It does not have to, but can be, the node name of the local machine. The local machine should be running a Web server (e.g. Apache, Sambar, etc.). Many ProxyPass lines referencing different machines may be added. And, yes, the trailing slashes make a difference but aren't always required.

Now stop, and restart Apache. Machines differ, but the following commands are likely to be the ones you need:

```
/etc/rc.d/init.d/httnpd stop

/etc/rc.d/init.d/httpd start
```

If you get errors, double-check your Apache installation. If you compiled and installed a fresh version of Apache, make sure that it is the one that you are actually running and the one whose .conf files you are editing.

If Apache performed a clean start, you should now have access to the inner machine. If the outer machine's domain is "outer.com", and ProxyPass created /ws1/, and the inner machine has a top level index.html file (i.e., if on the inner machine you can access httpd://localhost/index.html), then the following URL should bring up index.html from your inner machine:

```
http://outer.com/ws1/index.html
```

If this doesn't work, check the access_log and error_log for Apache on both the inner and outer machines to find out why. The next step is to make a small addition to the inner machine’s http server. If you are not using Apache, you will have to discover the equivalent to the following command. In httpd.conf you will find a line that's likely to look exactly the following:

```
ScriptAlias /cgi-bin/ "/home/httpd/cgi-bin/"
```

Add a line that looks exactly like this one (don't get rid of the old one) but with your ProxyPass name prepended to /cgi-bin/, as in:

```
ScriptAlias /ws1/cgi-bin/ "/home/httpd/cgi-bin/"
```

This will make it so that calls for files in /cgi-bin/ or /ws1/cgi-bin/ will both actually use /home/httpd/cgi-bin/.

Stop and restart the inner machine's http server to make the change take effect.

Now get a webSurvent job running on the inner machine (i.e., make sure that you can run your webSurvent job on the inner machine using localhost as the domain).

In each .html file that your study uses, change the reference to cgi-bin so that it includes the machine's ProxyPass name. For example, if the local version was:

```
<FORM name="cfmclogin" METHOD="POST" ACTION="/cgibin/websrv81.cgi">
```

Then the new version should be:

```
<FORM name="cfmclogin" METHOD="POST" ACTION="/ws1/cgibin/websrv81.cgi">
```

Finally, in each .html file that contains NAME="CFMC" add a line similar to the following, but with the value set to match the ProxyPass name you are using:

```
<INPUT type="hidden" NAME="PROXYPASS" VALUE="/ws1/">
```

Your webSurvent study should now be accessible to the outside world.

If, for example, /home/httpd/html is the top-level directory that your inner http server serves documents from (Apache's default), and your webSurvent job's index.html was in this directory (it's probably not), then the following URL should access it:

http://outer.com/ws1/index.html

You will have to use the full-length URL from both outside and inside the LAN. If you have trouble, use "tail –f" on both access_log and error_log on both the inner and outer machines.

Troubleshooting ProxyPass

The two most common problems that arise are:

When the outer machine says it can't find /cgi-bin/websrv81.cgi, which probably means that you missed prepending the ProxyPass name somewhere, or you did not set the ProxyPass variable in your .html files.

The other problem may be that the inner machine says it can't find /ws1/ws1/cgi-bin/websrv81.cgi (i.e., ws1 appears twice). This usually means that the page was called from the inner machine without using the complete URL (i.e., one that includes the outer machine's name).

Watch your log files as you attempt to go through a questionnaire, and you should be able to troubleshoot any ProxyPass problems you encounter.

INDEX