#!/usr/bin/perl

#/////////////////////////////////////////////////////////////////////////////

=head2 populate\_test\_template

 Parameters : template, test\_description, testable\_statement

 Returns : string

 Description : Populates the template with the test description and testable statement

 and returns the completed test string.

=cut

sub populate\_test\_template ($$$) {

 my $template = $\_[0];

 # test for needed namespaces

 $xlink = index( $\_[2], 'xlink') > -1;

 $html = index( $\_[2], '<html:') > -1;

 # Only add namespaces when needed, otherise remove XLINK placeholder from the template

 if ( $xlink ) {

 if ($html) {

 $template =~ s/XLINK/xmlns:xlink="http:\/\/www.w3.org\/1999\/xlink" xmlns:html="http:\/\/www.w3.org\/1999\/html" /;

 } else {

 # only xlink namepsace needed

 $template =~ s/XLINK/xmlns:xlink="http:\/\/www.w3.org\/1999\/xlink" /;

 }

 }

 elsif ($html) {

 # html namespace needed

 $template =~ s/XLINK/xmlns:html="http:\/\/www.w3.org\/1999\/html" /;

 }

 else {

 # no additional namespaces needed remove placeholder

 $template =~ s/XLINK//;

 }

 $template =~ s/COMMENT/$\_[1]/;

 $template =~ s/TESTABLE\_STATEMENT/$\_[2]/;

 return $template;

}

#/////////////////////////////////////////////////////////////////////////////

=head2 write\_testfile

 Parameters : contents, directory, test\_name, suffix

 Returns : the filename of the test file

 Description : writes the contents to a file in directory, the name of the file

 is determined from test\_name and suffix is added

=cut

sub write\_testfile ($$$$) {

 my $contents = $\_[0];

 chdir $\_[1];

 my $filename = create\_filename( $\_[2], $\_[3] );

 open( my $fh, ">:encoding(UTF-8)", $filename )

 || die "can't open UTF-8 encoded $filename: $!";

 print $fh $contents;

 close $fh;

 return $filename;

}

#/////////////////////////////////////////////////////////////////////////////

=head2 create\_filename

 Parameters : testname, suffix

 Returns : filename (unused in cwd)

 Description : trys to use the filename + suffix, if it exists it adds integers to

 filename until an unused filename is found.

=cut

sub create\_filename($$) {

 my $name = create\_filename\_from\_testname( $\_[0] );

 my $iter = 0;

 my $filename = $name . $\_[1];

 # test whether the $filename is unused, if not add an integer to the end of the filename

 while ( -e $filename ) {

 $iter++;

 $filename = $name . $iter . $\_[1];

 }

 return $filename;

}

#/////////////////////////////////////////////////////////////////////////////

=head2 test\_type

 Parameters : wiki header line

 Returns : test type

 Description : pulls the test type by finding the word after Testable statements for.

=cut

sub test\_type($) {

 if (index($\_[0], 'not included in accessibility tree') > -1) {

 return 'not in accessibility tree';

 }

 my $ind = index( $\_[0], 'for' );

 my $type = substr( $\_[0], $ind + 3 );

 $ind = index( $type, "==" );

 if ( $ind > 0 ) {

 $type = substr( $type, 0, $ind );

 }

 $type =~ s/^\s+|\s+$//g;

 return $type;

}

#/////////////////////////////////////////////////////////////////////////////

=head2 create\_filename\_from\_testname

 Parameters : testname

 Returns : filename

 Description : converts a testname to a filename.

=cut

sub create\_filename\_from\_testname($) {

 my $test\_name = $\_[0];

 # replace whitespace with single underscore

 $test\_name =~ s/\s+\_\*/\_/g;

 # remove any non-word characters

 $test\_name =~ s/\W//g;

 # taking out word accessible to make names shorter

 $test\_name =~ s/\_\*accessible//;

 # taking out word for to make names shorter

 $test\_name =~ s/\_+for\_+/ /;

 # remove any trailing underscores

 $test\_name =~ s/\_+$//;

 # remove any leading underscores

 $test\_name =~ s/^\_+//;

 return $test\_name;

}

=head2 read\_testfile

 Parameters : filename

 Returns : hash of array of tests, key is test type (role, name, description)

 Description : gets the tests as a description [0] and a testable statement [1]

=cut

sub read\_testfile ($) {

 open( FH, '<:encoding(UTF-8)', $\_[0] ) or die "Could not open file '$\_[0]' $!";

 my @lines = <FH>;

 close FH or die "Cannot close $\_[0]: $!";

 my %alltests;

 my $tests;

 my $todos = 0;

 my $counter = 0;

 for ( $i = 0 ; $i < @lines ; $i++ ) {

 if ( $lines[$i] =~ /^\s\*==\s+/ && $lines[$i] =~ /\s+==$/ ) {

 my $type = test\_type( $lines[$i] );

 my @testtype = ();

 $alltests{$type} = \@testtype;

 $tests = \@testtype;

 $counter = 0;

 }

 if ( $lines[$i] =~ /^;\s\*\/\// ) {

 if ( $lines[ $i + 1 ] =~ /^\s\*if/ ) {

 my $tlines = $lines[ $i + 1 ];

 my $index = $i + 2;

 # if testline is multiple lines, gather up all lines with SVG

 if ( index( $tlines, 'then' ) == -1 ) {

 while ( $index < @lines && index( $tlines, 'then' ) == -1 ) {

 $tlines .= $lines[$index];

 $index++;

 }

# # pick up expected result if on next line

# if ($index + 1 < @lines) {

# $tlines = $lines[$index + 1];

# if (index($tlines, ';') == -1 && index($tlines, 'if') == -1 && index($tlines, '==') == -1) {

# print $tlines;

# }

# }

 }

 my @test = ( $lines[$i], $tlines );

 @$tests[$counter] = \@test;

 $counter++;

 }

 }

 }

 return %alltests;

}

=head2 get\_svg

 Parameters : testable statement

 Returns : svg

 Description : gets the svg to be placed in the template

=cut

sub get\_svg($) {

 # substring from the first < to the 'then' clause

 my $start = index( $\_[0], '<' );

 my $end = rindex( $\_[0], '>' ) + 1;

 return substr( $\_[0], $start, ( $end - $start ) );

}

=head2 get\_element

 Parameters : svg

 Returns : element (type) used in test

 Description : returns the element type with the id="test"

=cut

sub get\_element($) {

 # find id="test"

 my $id = index($\_[0], 'id="test"');

 if ($id == -1) {

 # not found, try a weaker test

 $id = index($\_[0], '"test"');

 }

 my $start = rindex($\_[0], '<', $id) + 1;

 my $end = index($\_[0], " ", $start);

 return substr($\_[0], $start, $end - $start);

}

=head2 get\_expected\_result

 Parameters : testable statement

 Returns : expected result

 Description : gets the expected result of the testable statement

=cut

sub get\_expected\_result($) {

 my $v = $\_[0];

 if (index($\_[0], 'NOAC') > -1) {

 return 'NOT IN A11Y TREE';

 }

 if (index($\_[0], 'BLANK') > -1) {

 return 'null or empty';

 }

 #substring the result

 my $then = rindex($\_[0], 'then');

 my $start = rindex( $\_[0], '=' );

 # we expect remaining results to be then xy z = result, but if no = grab from then

 if ($then > $start) {

 # if expected result is clipped off give the user the end of the string

 $start = $then;

 }

 return substr( $\_[0], $start );

}

=head2 clean\_for\_output

 Parameters : string to clean

 Returns : cleaned string

 Description : cleans a string to be put into a csv file.

=cut

sub clean\_for\_output($) {

 $string = $\_[0];

 # replace comma with a space

 $string =~ s/,+/ /og;

 # remove an equal sign

 $string =~ s/\s\*=\s\*//og;

 # remove quotes

 $string =~ s/\"\*\'\*//og;

 # remove a newline

 $string =~ s/\n\*//og;

 # remove excess whitespace

 $string =~ s/\s+/ /og;

 $string =~ s/^\s//o;

 $string =~ s/\s$//o;

 return $string;

}

=head2 add\_csv\_line

 Parameters : csvfilehandle, columns

 Returns : none

 Description : gets the expected result of the testable statement

=cut

sub add\_csv\_line($$$$$$$$) {

 my $fh = shift;

 my $count = 0;

 my $first = shift;

 my $clean\_value = clean\_for\_output($first);

 print $fh $clean\_value;

 foreach $value (@\_) {

 $clean\_value = clean\_for\_output($value);

 print $fh ',' . $clean\_value;

 }

 print $fh "\n";

}

=head2 guess\_section

 Parameters : type

 Returns : section URI

 Description : guesses the section of the spec for the test.

=cut

sub guess\_section($) {

 my $var = lc($\_[0]);

 if ($var eq "name" || $var eq "description") {

 return "https://rawgit.com/w3c/aria/master/svg-aam/svg-aam.html#mapping\_additional\_nd";

 }

 if ($var eq "role is none" || $var eq "role is presentation") {

 return "https://rawgit.com/w3c/aria/master/svg-aam/svg-aam.html#exclude\_elements";

 }

 if (index($var, "role")) {

 return "https://rawgit.com/w3c/aria/master/svg-aam/svg-aam.html#mapping\_role\_table";

 }

 if ($var eq "not in accessibility tree") {

 return "https://rawgit.com/w3c/aria/master/svg-aam/svg-aam.html#mapping\_role\_table";

 }

 return '';

}

=head2 add\_json\_test\_object

 Parameters : filehandle,

 Returns : none

 Description : gets the expected result of the testable statement

=cut

sub add\_json\_test\_object($$$$$$) {

 my $fh = shift;

 my $json = "\n {\n\t\"testing\":\"";

 $json .= clean\_for\_output( $\_[0] );

 $json .= "\",\n\t\"element\":\"";

 $json .= clean\_for\_output( $\_[1] );

 $json .= "\",\n\t\"attributes\":\"";

 $json .= clean\_for\_output( $\_[2] );

 $json .= "\",\n\t\"file\":\"";

 $json .= clean\_for\_output( $\_[3] );

 $json .= "\",\n\t\"expected\":\"";

 $json .= clean\_for\_output( $\_[4] );

 $json .= "\"\n }";

 print $fh $json;

}

# main program

# directories and file names

# where the wiki markup is

my $wiki\_markup\_filename = 'c:\\bob\\wiki.txt';

#directories to write tests and database files to

$test\_dir = "c:\\test\\";

$html\_dir = $test\_dir . "html\\";

$svg\_dir = $test\_dir . "svg\\";

# database filenames

my $csv\_filename = $test\_dir . "svg\_html.csv";

my $json\_filename = $test\_dir . "svghtml.json";

my $csv\_svg\_filename = $test\_dir . "svg\_svg.csv";

my $json\_svg\_filename = $test\_dir . "svgsvg.json";

# create the test directories if they don't exist

mkdir($test\_dir) unless ( -d $test\_dir );

mkdir($html\_dir) unless ( -d $html\_dir );

mkdir($svg\_dir) unless ( -d $svg\_dir );

# alltests is a hash with all the files from the wiki markup

my %alltests = read\_testfile($wiki\_markup\_filename);

# template for HTML test files

my $html\_template = '<!DOCTYPE html>

<html>

 <head>

 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

 <title> COMMENT </title>

 </head>

 <body>

 <svg xmlns="http://www.w3.org/2000/svg" XLINK >

 TESTABLE\_STATEMENT

 </svg>

 </body>

</html>';

# template for SVG test files

my $svg\_template = '<svg xmlns="http://www.w3.org/2000/svg" XLINK >

 TESTABLE\_STATEMENT

</svg>';

#open the database of tests files (csv and json)

open( my $csv\_html, ">:encoding(UTF-8)", $csv\_filename )

 || die "can't open UTF-8 encoded $csv\_filename: $!";

open( my $json\_html, ">:encoding(UTF-8)", $json\_filename )

 || die "can't open UTF-8 encoded $json\_filename: $!";

 open( my $csv\_svg, ">:encoding(UTF-8)", $csv\_svg\_filename )

 || die "can't open UTF-8 encoded $csv\_filename: $!";

open( my $json\_svg, ">:encoding(UTF-8)", $json\_svg\_filename )

 || die "can't open UTF-8 encoded $json\_filename: $!";

# setup the csv files, add column headers

add\_csv\_line($csv\_html, "TESTING", "ELEMENT", "ATTRIBUTES", "FILE", "EXPECTED VALUE", "DOC SECTION", "API" );

add\_csv\_line($csv\_svg, "TESTING", "ELEMENT", "ATTRIBUTES", "FILE", "EXPECTED VALUE", "DOC SECTION", "API" );

# start the json file (open an array)

print $json\_html "[\n";

print $json\_svg "[\n";

my $counter = 0;

# main loop, building test files and database files

foreach my $key ( keys %alltests ) {

 my $tests = $alltests{$key};

 # simplify the type for database files (description, name, role)

 my $type = $key;

 $type =~ s/calculation//o;

 $type =~ s/Role/role/o;

 my $section = guess\_section($type);

 my $typeCount = 0;

 # make the testfiles and populate the database files

 foreach my $test (@{$alltests{$key}}) {

 my $svg = get\_svg( $$test[1] );

 my $element = get\_element($svg);

 my $comment = $$test[0];

 $comment =~ s/;\s\*\/\///;

 # make the HTML test file

 my $h = populate\_test\_template( $html\_template, $comment, $svg );

 my $html\_filename = write\_testfile( $h, $html\_dir, $$test[0], ".html" );

 # make the SVG test file

 my $s = populate\_test\_template( $svg\_template, $comment, $svg );

 my $svg\_filename = write\_testfile( $s, $svg\_dir, $$test[0], ".svg" );

 # add the test to the csv (HTML) database file

 my $expected = get\_expected\_result( $$test[1] );

 add\_csv\_line( $csv\_html, $type, $element, $comment, $html\_filename, $expected,$section,'' );

 add\_csv\_line( $csv\_svg, $type, $element, $comment, $svg\_filename, $expected,$section,'' );

 # add the test to the JSON (HTML) database file

 if ( $counter > 0 ) {

 # don't have a comma before the first object in the array

 print $json\_html ',';

 print $json\_svg ',';

 }

 add\_json\_test\_object( $json\_html, $type, $element, $comment, $html\_filename, $expected );

 add\_json\_test\_object( $json\_svg, $type, $element, $comment, $svg\_filename, $expected );

 $counter++;

 $typeCount++;

 }

 print $typeCount." tests for ".$type."\n";

}

print $counter." tests were written \n";

# finish and close JSON database files

print $json\_html "\n]";

close $json\_html;

print $json\_svg "\n]";

close $json\_svg;

# close the csv database file

close $csv\_html;

close $csv\_svg;

print 'program exited normally';