Xdg-utils Test Specification

NOTE: This draft will be replaced with a format that allows test code to be tied to the test specification in a way that is easier to maintain.

General assumptions:

1. If any dependency fails, return a test error result, not a failure.
2. Every test has the implicit dependency that the utility being tested is in the $PATH. Generally, this is tested by the first unit test for each utility.
3. Each test starts in an identical environment.
4. Any files or settings created or modified by the test will be removed or restored after the test is complete.
5. Test with a USER INTERACTIVE dependency are labeled as such because it is not clear how to automate what is being tested. We seek a method to perform this automation.

Software Framework:

Automatable tests will be written to run as standalone tests, or as part of 'make test'. Additionally, the tests will be written to be easily wrapped into the Tet framework from the Open Group (http://tetworks.opengroup.org) so that the LSB project can use the tests without too much extra work.

Filename tests

Many xdg-utils operate on local files. Many different possibilities for non-traditional filenames exist. Each utility that operates on a file name should accept filenames of the following type. Each test that specifies a file will be run for each of these types of filenames. An exception may be made for USER INTERACTIVE tests until such tests are automated.

1. Base case: “normal” unix paths with ASCII non-whitespace characters.
2. symlinks
3. paths that contain spaces, tabs or newlines
4. paths that contain unicode characters
5. paths that contain 8-bit non-unicode characters
6. relative vs. absolute paths

URL tests

In cases where a URL is being translated from URL space to local filename space (or vice versa), the test will ensure that encoded or decoded correctly.

1. Base case: “normal” URLs that contain no spaces or encoded characters.
2. file:/// URLs – verify that encoded text is correctly matched to the local file.
3. URLs that contain encoded whitespace characters.
4. URLs that contain encoded Unicode characters.

Generic unit tests:
These tests will be run for each utility, where XDG-UTIL is the name of the utility we are testing. These tests are identical for each utility.

1. **Test**: no argument test
   Purpose: Ensure that XDG-UTIL exits with an error and usage message when called without required arguments.
   Description: Run the utility with no arguments and check the output.
   Dependencies:
   Test sequence:
   1. Initialize
      1. Clear DISPLAY
   2. Execute “XDG-UTIL”
      1. Expect exit 1.
      2. Expect no STDOUT
      3. Expect STDERR

   Notes:
2. **Test**: bogus argument test
   Purpose: Ensure utility can handle bad arguments
   Description: Run the utility with '--foobar-bogus-argument' and verify an error message is printed to STDOUT
   Dependencies:
   Test Sequence:
   1. Initialize
      1. Clear DISPLAY
   2. Execute “XDG-UTIL --foobar-bogus-argument'”
      1. Expect exit 1.
      2. Expect no STDOUT
      3. Expect STDERR

   Notes:
3. **Test**: help tests
   Purpose: ensure all of the help, version and manual options work.
   Description: Run the utility with each of {--help,--manual,--version} and verify that output appears on STDOUT with exit 0.
   Dependencies:
   Test Sequence:
   1. Initialize
      1. Clear DISPLAY
   2. Execute each of {“XDG-UTIL –help”, “XDG-UTIL –manual”, “XDG-UTIL –version”}
      1. Expect exit 0.
      2. Expect STDOUT
      3. Expect no STDERR

   Notes:

Unit Test Descriptions:

1. **Utility**: xdg-mime
    1. Generic tests.
    2. **Test**: text mime type
       Purpose: Verify that the basic "text/plain" type is working.
       Description: Run the utility on a generic text file and see that "text/plain" is returned. Note that we are testing file rules described on p. 1.
       Dependencies:
       Test sequence:
        1. Initialize
            1. Clear DISPLAY
            2. Create <file>.txt as an ASCII file.
        2. Execute "xdg-mime query filetype <file>.txt"
            1. Expect exit 0
            2. Expect STDOUT to match "text/plain" exactly.
            3. Expect no STDERR

       Notes:
    3. **Test**: user mime install test
       Purpose: Verify that new mime types are installed correcty.
       Description: Install a test mime type, and verify that it is installed correctly.
       Dependencies:
        1. testmime.xml is provided with the test suite, describing an extension TESTEXT, type TESTMIMETYPE.

       Test sequence:
        1. Initialize
            1. Clear DISPLAY
            2. Create foo.TESTEXT
        2. Execute each of "xdg-mime install --user testmime.xml"
            1. Expect exit code 0
            2. Expect no STDERR
            3. Expect no STDOUT
        3. Execute "xdg-mime query filetype foo.TESTEXT"
            1. Expect exit code 0
            2. Expect STDOUT to match TESTMIMETYPE, as defined by testmime.xml
            3. Expect no STDERR
    4. **Test**: system mime install test
       Purpose: Verify that new mime types are installed correcty.
       Description: Install a test mime type, and verify that it is installed correctly.
       Dependencies:
        1. testmime.xml is provided with the test suite, describing an extension TESTEXT, type TESTMIMETYPE.
        2. Root access.

       Test sequence:
        1. Initialize
            1. Clear DISPLAY
            2. Create foo.TESTEXT
        2. Execute each of "xdg-mime install --system testmime.xml"
            1. Expect exit code 0

    2. Expect no STDERR
    3. Expect no STDOUT
3. Execute “xdg-mime query filetype foo.TESTEXT”
    1. Expect exit code 0
    2. Expect STDOUT to match TESTMIMETYPE, as defined by testmime.xml
    3. Expect no STDERR

Notes/Questions:

5. Test: user mime uninstall test

Purpose: Verify that mime types are correctly removed from the database.

Description: Install a mime type, then remove it.

Dependencies:

1. Test 2.2 (install mime type) is PASS

Test sequence:

1. Initialize
    1. Clear DISPLAY
    2. Install testmime.xml (see 2.2)
2. Execute “xdg-mime uninstall –user testmime.xml”
    1. Expect exit code 0
    2. Expect no STDOUT
    3. Expect no STDERR
3. Execute “xdg-mime query filetype foo.TESTEXT”
    1. Expect exit code 0 (??)
    2. Expect STDOUT not to contain TESTMIMETYPE.
    3. Expect no STDERR

Notes/Questions:

6. Test: system mime uninstall test

Purpose: Verify that mime types are correctly removed from the database.

Description: Install a mime type, then remove it.

Dependencies:

1. Test 2.2 (install mime type) is PASS
2. root access

Test sequence:

1. Initialize
    1. Clear DISPLAY
    2. Install testmime.xml (see 2.2)
2. Execute “xdg-mime uninstall –system testmime.xml”
    1. Expect exit code 0
    2. Expect no STDOUT
    3. Expect no STDERR
3. Execute “xdg-mime query filetype foo.TESTEXT”
    1. Expect exit code 0 (??)
    2. Expect STDOUT not to contain TESTMIMETYPE.
    3. Expect no STDERR

Notes/Questions:

7. **Test**: double install

Purpose: Verify that installing over an already installed mime type does not affect behavior.

Description:

Dependencies:

1. testmime.xml is provided with the test suite, describing an extension TESTEXT, type TESTMIMETYPE.

Test sequence:

1. Initialize
    1. Clear DISPLAY
    2. Create foo.TESTEXT
    3. Install testmime.xml
2. Execute “xdg-mime install --user testmime.xml”
    1. Expect exit code 0
    2. (permit warnings on STDERR and STDOUT)
3. Execute “xdg-mime query filetype foo.TESTEXT”
    1. Expect exit code 0
    2. Expect STDOUT to match TESTMIMETYPE, as defined by testmime.xml
    3. Expect no STDERR

8. **Test**: uninstall not installed

Purpose: Verify removing a not-installed mime type does not generate errors

Description:

Dependencies:

1. Test 2.2 (install mime type) is PASS

Test sequence:

1. Initialize
    1. Clear DISPLAY
2. Execute “xdg-mime uninstall –user testmime.xml”
    1. Expect exit code 0
    2. (permit warnings on STDERR or STDOUT)
3. Execute “xdg-mime query filetype foo.TESTEXT”
    1. Expect exit code 0
    2. Expect STDOUT not to contain TESTMIMETYPE.
    3. Expect no STDERR

Notes/Questions:

9. **Test**: query non-existant file (bug 7123 case 1)

Purpose: Verify exit code 2 when queried with a file that does not exist.

Description:

Dependencies:

Test sequence:

1. Initialize
    1. Clear DISPLAY
2. Execute “xdg-mime non-existing-file.bar”
    1. Expect exit code 2
    2. Expect no STDOUT.
    3. Expect STDERR

Notes/Questions:

10. **Test**: query foo.bar containing text “foobar”.

Purpose: Verify we get 'text/plain' for an unknown test file.

Description:

Dependencies:

1. .bar is not a registered file type.

Test sequence:

1. Initialize
    1. Clear DISPLAY
    2. Create 'foo.bar' with text 'foobar'
2. Execute “xdg-mime query filetype foo.bar”
    1. Expect exit code 0
    2. Expect STDOUT to match “text/plain”.
    3. Expect STDERR

Notes/Questions:

2. Utility: xdg-open
    1. Generic Tests
    2. **Test**: Open test (manual)
    Purpose: Verify that each file and URL type (see p.1) is opened correctly.
    Description: Open a file/URL using each protocol. Manually verify that the windows open as expected. (Expectations will vary.)
    Dependencies:
        1. USER INTERACTIVE
        2. Xdg-mime –defapp "foo.html" returns the users's preferred browser.
    Test sequence: (for each of {<filesystem>,file://, http://,https://, ftp://})
        1. Execute "xdg-mime –defapp foo.html" and record result R3_2
        2. Execute "which R3_2"
            1. Expect exit 0 or return NORESULT
        3. Execute "xdg-open <url>"
            1. Expect exit 0
        4. Query user: "Did R3_2 open?"
            1. Expect YES
        5. Query user: "Does the <some description> match?"
            1. Expect YES
    Notes: We cannot expect no STDERR/STDOUT since some environments (eg SUSE/Gnome/Firefox) print debug messages to the console.
    3. **Test**: URL open test (automatic)
    Purpose: Verify the URL is opened as expected.
    Description: Call xdg-open, then examine the server's access log to see if the correct HTTP/HTTPS/FTP request was made.
    Dependencies:
        1. Lightweight HTTP/FTP servers.
        2. Xdg-mime –defapp "foo.html" returns the users's preferred browser.
        3. Running X server
    Test sequence:
        1. Initialize:
            1. Create <file>
        2. Record R4_2 = "xdg-mime –defapp foo.html"
        3. Execute "xdg-open http://localhost:PORT/<file>"
            1. Expect exit 0
            2. Expect no STDERR
            3. Expect no STDOUT
            4. Expect to find <file> in the server's access log .
            5. Find the USER_AGENT string on the line for <file>. Expect it to match based on R4_2.
    Notes:
        1. It may be possible to test the FTP protocol in a similar way.
        2. For <filesystem> and file:// it may be possible to use something like inotify/atime to see when a file was accessed.
    4. **Test**: directory open test
    Purpose:Verify that a directory browser opens correctly.
    Description: xdg-open <dir> and verify that the user's directory browser opens.
    Dependencies:

1. USER INTERACTIVE
2. Running X server

Test sequence:

1. Initialize:
    1. Create <dir>
2. Execute “xdg-open <dir>”
    1. Expect exit 0
    2. Expect no STDERR
    3. Expect no STDOUT
3. Ask user “Did <dir> open in a directory browser?”
    1. Expect YES

Notes:

3. Utility: xdg-email
    1. Generic tests.
    2. **Test**: mail to
    Purpose: Verify the most basic use case.
    Description: Open a mailto: URI that contains only an address.
    Dependencies:
        1. USER INTERACTIVE
        2. Running X server
        3. Email client is sufficiently configured to send email.(eg. no “Setup Wizard” will appear)
    Test Sequence:
        1. Execute “xdg-email 'mailto:user@test.org'”
            1. Expect exit 0
        2. Ask user “Did email window open with correct email address?”
            1. Expect YES
    Notes:
    3. Test: basic compose mail
    Purpose: Make sure that the email composed correctly.
    Description: Run xdg-email and verify that the email shows up correctly.
    Dependencies:
        1. USER INTERACTIVE
        2. Running X server
        3. Email client is sufficiently configured to send email.(eg. no “Setup Wizard” will appear)
    Test sequence:
        1. Initialize
            1. Create <attachfile>
        2. Execute “xdg-email –cc 'testCC@user.org' –bcc 'testBCC@user.org' –subject 'Test Subject' –body 'Body test' –attach <attachfile> 'testTO@user.org'
            1. Expect exit 0
        3. Ask user to verify fields filled. For each argument, present the user with the argument string and ask “Does <field> match <data>?”
            1. Expect YES
    Notes: This could be automated by simply asking the user to click send and having the mail verified by some automated tool.
    4. Test: multiple address compose
    Purpose: Verify multiple addresses show up correctly.
    Description: Run xdg-email and verify addresses show up correctly.
    Dependencies:
        1. USER INTERACTIVE
        2. Running X server
        3. Email client is sufficiently configured to send email.(eg. no “Setup Wizard” will appear)
    Test sequence:
        1. Execute “xdg-email –cc 'testCC1@user.org' –cc 'testCC2@user.org –bcc 'testBCC1@user.org' –bcc 'testBCC2@user.org' –subject 'Test Subject' –body 'Body test' 'testTO1@user.org' 'testTO2@user.org'
            1. Expect exit 0
        2. Ask user to verify fields filled. For each argument, present the user with the argument string and ask “Does <field> match <data>?”

1. Expect YES

Notes: This could be automated by simply asking the user to click send and having the mail verified by some automated tool.

5. **Test**: mailto URI parse – single address
   Purpose: Verify mailto: is parsed correctly
   Description: Run xdg-email and verify addresses show up correctly.
   Dependencies:
   1. USER INTERACTIVE
   2. Running X server
   3. Email client is sufficiently configured to send email.(eg. no "Setup Wizard" will appear)

   Test sequence:
   1. Execute "xdg-email mailto:testTO@user.org?subject=Test%20Subject&cc=testCC@user.org"
      1. Expect exit 0
   2. Ask user to verify fields filled. For each argument, present the user with the argument string and ask "Does <field> match <data>?"
      1. Expect YES

   Notes: This could be automated by simply asking the user to click send and having the mail verified by some automated tool.
6. **Test**: mailto URI parse – multiple addresses
   Purpose: Verify mailto: is parsed correctly with mutliple addresses
   Description: Run xdg-email and verify addresses show up correctly.
   Dependencies:
   1. USER INTERACTIVE
   2. Running X server
   3. Email client is sufficiently configured to send email.(eg. no "Setup Wizard" will appear)

   Test sequence:
   1. Execute "xdg-email mailto:testTO@user.org%2C%20testTO2@user.org?subject=Test%20Subject&cc=testCC@user.org&testCC2@user.org"
      1. Expect exit 0
   2. Ask user to verify fields filled. For each argument, present the user with the argument string and ask "Does <field> match <data>?"
      1. Expect YES

   Notes: This could be automated by simply asking the user to click send and having the mail verified by some automated tool.
7. **Test**: mailto URI parse – single address plus arguments
   Purpose: Verify mailto: is parsed correctly
   Description: Run xdg-email and verify addresses show up correctly.
   Dependencies:
   1. USER INTERACTIVE
   2. Running X server
   3. Email client is sufficiently configured to send email.(eg. no "Setup Wizard" will appear)

   Test sequence:
   1. Execute "xdg-email –cc testCC2@user.org –attach <file> --body "My body" –bcc testBCC@user.org mailto:testTO@user.org?subject=Test%20Subject&cc=testCC@user.org "
      1. Expect exit 0
   2. Ask user to verify fields filled. For each argument, present the user with the argument string and ask "Does <field> match <data>?"

1. Expect YES

Notes: This could be automated by simply asking the user to click send and having the mail verified by some automated tool.

8. **Test**: weird filenames
   Purpose: ensure that filenames can be attached, for all filenames described on p. 1
   Description: Run xdg-email –attach for various files. Check that they get attached by the MUA.
   Dependencies:
   1. USER INTERACTIVE
   2. Running X server
   3. Email client is sufficiently configured to send email.(eg. no "Setup Wizard" will appear)

   Test sequence:
   1. Execute "xdg-email –cc testCC2@user.org –attach <file> --body "My body" –bcc testBCC@user.org mailto:testTO@user.org?subject=Test%20Subject&cc=testCC@user.org "
      1. Expect exit 0
   2. Ask user to verify fields filled. For each argument, present the user with the argument string and ask "Does <field> match <data>?"
      1. Expect YES

   Notes:
9. **Test**: nonUTF8 locales
   Purpose: Verify that –utf8 works
   Description: Submit a subject that is UTF8, while the current locale is NOT UTF8.
   Dependencies:
   1. USER INTERACTIVE
   2. Running X server
   3. Email client is sufficiently configured to send email.(eg. no "Setup Wizard" will appear)
   4. Environment capable of being set to a locale other than UTF8

   Test sequence:
   1. Initialize
      1. Set locale to something other than UTF8
   2. Execute "xdg-email –utf8 –subject <some UTF8 string> test@user.org"
      1. Expect exit 0
   3. Ask user: "Does subject match <some UTF8 string> ?
      1. Expect YES

   Notes:
10. **Test**: refuse bogus attach
    Purpose: Ensure that malicious programs cannot put arguments to xdg-email into mailto: URLs, eg (xdg-email 'mailto:user@org –attach /etc/passwd')
    Description:
    Dependencies:
    1. USER INTERACTIVE
    2. Running X server
    3. Email client is sufficiently configured to send email.(eg. no "Setup Wizard" will appear)

    Test sequence:
    1. Execute "xdg-email mailto:user@test.org –attach <file>"
    2. Ask user "Does email have an attachment?"
       1. Expect NO.

    Notes:
11. Test: body in mailto

Purpose: verify mailto urls support having body text including properly encoded newlines
Description:
Dependencies:
1. USER INTERACTIVE
2. Running X server
3. Email client is sufficiently configured to send email.(eg. no “Setup Wizard” will appear)

Test Sequence:
1. Execute “xdg-email mailto:user@test.org?body=some%20text%<newline>some%20text%20on%20line%202”
    1. expect exit 0
    2. expect no STDERR
    3. expect no STDOUT
2. Ask user does the text “some text” appear on the first line and “some text on line 2” appear on the second line of the body?
    1. Expect YES.

Notes:

12. Test: no attachment in URL

Purpose: ensure xdg-email blocks URLs that contain attachments
Description:
Dependencies:
1. USER INTERACTINVE
2. Running X server
3. Email client is sufficiently configured to send email.(eg. no “Setup Wizard” will appear)

Test Sequence:
1. Initialize
    1. Create FILE
2. Execute “xdg-email mailto:test@user.org?attach=FILE”
    1. Expect exit 2
    2. Expect STDERR
    3. Expect no STDOUT
3. Ask user “Does the email contain an attachmetn?”
    1. Expect NO

Notes:

4. Utility: xdg-icon-resource
    1. Generic tests.
    2. **Test**: user icon install test
       Purpose: Verify that icons get installed.
       Description: Install an icon and verify that it is present in the icon path.
       Dependencies:
        1. test icon TESTICON
        2. $ICONPATH is defined by the standards at freedesktop.org. All icons should be installed here.
        3. diff utility
        4. find utility

       Test sequence: (run for each of .png and .svg)
        1. Initialize
            1. Clear DISPLAY
        2. Execute “xdg-icon-resource nstall TESTICON --user”
            1. Expect exit code 0
            2. Expect no STDOUT
            3. Expect no STDERR
        3. Execute “find $ICONPATH -name '*TESTICON*'
            1. Expect one instance of .icon
            2. Expect one instance of .png or .svg
            3. Expect ICONFOUND.icon to match TESTICON.icon exactly
            4. Expect ICONFOUND.png to match TESTICON.png exactly (or .svg)

       Notes: It could make sense to add a “--find” argument to xdg-icon which returns the path of the quried icon. (eg, Implement what is given in pseudocode in the icon theme spec. http://standards.freedesktop.org/icon-theme-spec/icon-theme-spec-latest.html)
    3. **Test**: user icon remove (run for each of .png and .svg)
       Purpose: Verify that an icon is removed correctly
       Description: Remove an icon and verify that it is no longer in the $ICONPATH
       Dependencies:
        1. Test 5.2 (icon install) is PASS
        2. $ICONPATH is defined by the standards at freedesktop.org. All icons should be installed here.
        3. find utility

       Test sequence:
        1. Initialize
            1. Clear DISPLAY
            2. Install TESTICON (See 5.2)
        2. Execute “xdg-icon-resource unistall TESTICON --user”
            1. Expect exit 0
            2. Expect no STDOUT
            3. Expect no STDERR
        3. Execute “find $ICONPATH -name '*TESTICON*'”
            1. Expect no STDOUT

       Notes: Again, a “--find” argument would be useful here.
    4. Test: system icon install (see 2)
       Identical to user install except that we need root access and –system replaces –user.
    5. Test: system icon uninstall (see 3)
       Identical to user install except that we need root access and –system replaces –user.
    6. Test: icon install no user/system

Purpose: verify an error is generated when the required –system or –user argument is not present.
Description:
Test Sequence:

1. Initialize
    1. Clear display
2. Execute “xdg-icon-resource install TESTICON”
    1. Expect exit 1
    2. Expect STDERR
    3. Expect no STDOUT
3. Execute “find ICONPATH –name TESTICON”
    1. Expect no STDOUT
    2. Expect no STDERR

7. Test: uninstall not installed icon
    1. Initialize
        1. Clear display
        2. Verify TESTICON is not installed.
    2. Execute “xdg-icon-resource uninstall TESTICON –user
        1. Expect exit 4
        2. Expect STDERR
8. Test: install duplicate icon
    Test procedure:
    1. Initialize
        1. Clear display
        2. execute “find ICONPATH –name TESTICON”
            1. save STDOUT
    2. Execute “xdg-icon-resource install TESTICON”
        1. expect exit 0
        2. expect no STDERR
        3. (allow for warning on STDOUT)
    3. Execute “find ICONPATH –name TESTICON”
        1. expect a match to the result saved in the initialization step.
9. Test: forceupdate user
    Return UNTESTED – how do we test?
10. Test: forceupdate system
    Return UNTESTED – how do we test?
11. Test: forceupdate no user/sysem
    Return UNTESTED – how do we test?
    Note: it is not clear how to verify that forceupdate does anything.
12. Test: context (leave for future release – need to understand use case)
    Test sequence: Return UNTESTED
13. **Test**: Themes
    Purpose: Test the –theme install option.
    Description:
    Dependencies:
    Test sequence: return UNTESTED
    Notes: Best to leave this test for a future release.
    Need more details about themes to test them:
    1. How are themes created?

2. What constitutes a theme?
3. How can we tell what the current theme is?

5. Utility: xdg-desktop-icon
    1. Generic tests.
    2. **Test**: Desktop Install
    Purpose: Verify that a .desktop file is installed correctly.
    Description: Run the utility with a .desktop file that specifies an icon and application. Verify that the icon is correct and that the application specified is launched.
    Dependencies:
        1. USER INTERACTIVE
        2. Test 5.2 (icon install) is PASS
        3. An executable script that outputs a temporary file RESULTFILE
        4. A .desktop file DESKFILE describing the test scenario.
    Test sequence:
        1. Initialize
            1. Clear DISPLAY
        2. Execute “xdg-desktop-icon –install DESKFILE”
            1. Expect exit 0
            2. Expect no STDOUT
            3. Expect no STDERR
        3. Query user: “Does does icon appear?”
            1. Expect YES
        4. Instruct user: “Double click icon.” Wait for completion indiction.
            1. Expect RESULTFILE to be present.
    Notes:
    3. **Test**: Desktop Remove
    Purpose: Verify that a .desktop file is installed correctly
    Description: Run the utility with the .desktop file suppied in 6.2 (desktop install), and verify that the entry is removed.
    Dependencies:
        1. USER INTERACTIVE
        2. Test 6.2 (desktop install) is PASS
        3. A .desktop file DESKFILE describing the test scenario, as in test 6.2 (desktop install)
    Test sequence:
        1. Initilaize:
            1. Clear DISPLAY
            2. Install DESKFILE (see 6.2)
        2. Execute “xdg-desktop-icon –remove DESKFILE”
            1. Expect exit 0
            2. Expect STDOUT
            3. Expect STDERR
        3. Query user: “Does icon appear?”
            1. Expect NO
    Notes:
    4. **Test**: Install duplicate file.
    Purpose: Verify that a .desktop file is installed correctly.
    Description: Run the utility with a .desktop file that specifies an icon and application. Verify that the icon is correct and that the application specified is launched.
    Dependencies:
        1. USER INTERACTIVE

2. Test 5.2 (icon install) is PASS
3. An executable script that outputs a temporary file RESULTFILE
4. A .desktop file DESKFILE describing the test scenario.

Test sequence:

1. Initialize
    1. Clear DISPLAY
    2. Install DESKFILE
2. Execute “xdg-desktop-icon –install DESKFILE”
    1. Expect exit 0
    2. (permit warnings on STDERR/STDOUT)
3. Ask user: “Does does icon appear exactly once?”
    1. Expect YES
4. Instruct user: “Double click icon.” Wait for completion indiction.
    1. Expect RESULTFILE to be present.

Notes:

5. **Test**: Remove non existing file.

Purpose: Verify that removing a non-existant desktop file will not cause errors.

Description:

Dependencies:

1. USER INTERACTIVE
2. A .desktop file DESKFILE describing the test scenario, as in test 6.2 (desktop install)

Test sequence:

1. Initilaize:
    1. Clear DISPLAY
2. Execute “xdg-desktop-icon –remove DESKFILE”
    1. Expect exit 0
    2. (permit a warning on STDERR/STDOUT)
3. Query user: “Does icon appear?”
    1. Expect NO

Notes:

6. **Utility**: xdg-desktop-menu
   1. **Test**: Generic tests.
   2. **Test**: install submenu
      Purpose: verify menus are created correctly
      Description: Create a submenu with a supplied .menu and .directory file. Ask the user if menus were created correctly.
      Dependencies:
      1. USER INTERACTIVE
      2. Test supplied t7-2.menu and t7-2.directory files.

      Test sequence:
      1. Initialize:
         1. Clear DISPLAY
      2. Execute "xdg-desktop-menu –install –user t7-2.menu"
         1. Expect exit 0
         2. Expect no STDOUT
         3. Expect no STDERR
      3. Execute "xdg-desktop-menu –install –user t7-2.directory"
         1. Expect exit 0
         2. Expect no STDOUT
         3. Expect no STDERR
      4. Ask user if empty menu appears in location TBD.
         1. Expect YES

      Notes/Questions:
   3. **Test**: install menu item
      Purpose: verify menu items are created correctly.
      Description: Create a menu item with a supplied .desktop file. Ask the user to select this menu item.
      Dependencies:
      1. USER INTERACTIVE
      2. Test supplied t7-3.desktop
      3. Test supplied executable script that is associated with the .desktop file. Script will create a file in a defined location.
      4. Test 7.2 (create submenu) is PASS

      Test sequence:
      1. Initialize:
         1. Clear DISPLAY
         2. Create target submenu (see 7.2)
      2. Execute "xdg-desktop-menu –install –user t7-3.desktop"
         1. Expect exit 0
         2. Expect no STDERR
         3. Expect no STDOUT
      3. Ask user "Does menu item T7-3 appear in submenu T7-2?"
         1. Expect YES
      4. Instruct user "Select menu item T7-2/T7-3"
         1. Expect file to be created as specified by the script.

      Notes:
   4. **Test**: Menu item remove
      Purpose: Verify that items are removed from a menu item correctly.

Description: Remove an item from a menu and ask the user to verify that it has been removed.
Dependencies:
1. USER INTERACTIVE
2. Tests 7.2 (create submenu) and 7.3 (create menu item) are PASS

Test sequence:
1. Initialize:
    1. Clear DISPLAY
    2. Create target submenu (see 7.2)
    3. Create target menuitem in submenu (see 7.3)
2. Execute "xdg-desktop-menu –uninstall –user ITEM.desktop"
    1. Expect exit 0
    2. Expect no STDOUT
    3. Expect no STDERR
3. Ask user "Does menu item appear in submenu "
    1. Expect NO

Notes:

5. Test: Submenu remove
Purpose: Verify that menus are removed correctly.
Description: Execute tests 7.2 then remove the menu. Ask the user to verify that the menu is no longer present.
Dependencies:
1. USER INTERACTIVE
2. Test 7.2 is PASS

Test sequence:
1. Initialize:
    1. Clear DISPLAY
    2. Create target submenu (see 7.2)
2. Execute "xdg-desktop-menu –uninstall –user SUB.menu"
    1. Expect exit 0
    2. Expect no STDOUT
    3. Expect no STDERR
3. Execute "xdg-desktop-menu –uninstall –user SUB.directory"
    1. Expect exit 0
    2. Expect no STDOUT
    3. Expect no STDERR
4. Ask user "Does menu SUB appear?"
    1. Expect NO

Notes/Questions:

6. **Test**: Install existing menu item.
Purpose: Verify duplicate menus are not installed.
Description:
Dependencies:
1. USER INTERACTIVE
2. Test supplied t7-3.desktop
3. Test supplied executable script that is associated with the .desktop file. Script will create a file in a defined location.
4. Test 7.2 (create submenu) is PASS

Test sequence:

1. Initialize:
    1. Clear DISPLAY
    2. Create target submenu (see 7.2)
    3. Install t7-3.desktop.
2. Execute “xdg-desktop-menu –install –user t7-3.desktop”
    1. Expect exit 0
    2. Expect no STDERR
    3. Expect no STDOUT
3. Ask user “Does menu item T7-3 appear in submenu T7-2?”
    1. Expect YES
4. Instruct user “Select menu item T7-2/T7-3”
    1. Expect file to be created as specified by the script.

Notes:

7. **Test**: Install existing menu.
   Purpose:
   Description:
   Dependencies:
   1. USER INTERACTIVE
   2. Test supplied t7-2.menu and t7-2.directory files.
   3. Install tests PASS.

   Test sequence:
   1. Initialize:
       1. Clear DISPLAY
       2. Install MENU
   2. Execute “xdg-desktop-menu –install –user MENU.menu”
       1. Expect exit 0
       2. Expect no STDOUT
       3. Expect no STDERR
   3. Execute “xdg-desktop-menu –install –user t MENU.directory”
       1. Expect exit 0
       2. Expect no STDOUT
       3. Expect no STDERR
   4. Ask user if exactly one (not two) menu appears in a location TBD.
       1. Expect YES

   Notes:
8. **Test**: Remove non-existant menu item
   Purpose: Verify removing a non-esiting menu item does not generate extra errors.
   Description:
   Dependencies:
   1. USER INTERACTIVE
   2. Tests 7.2 (create submenu) and 7.3 (create menu item) are PASS

   Test sequence:
   1. Initialize:
       1. Clear DISPLAY
       2. Create target submenu (see 7.2)
       3. Create target menuitem in submenu (see 7.3)
   2. Execute “xdg-desktop-menu –uninstall –user ITEM.desktop”
       1. Expect exit 0

    2. (Note: we will permit warnings on STDERR/STDOUT)
3. Ask user “Does menu item appear in submenu ”
    1. Expect NO

Notes:

9. **Test**: Remove non-existant menu

Purpose: Verify removing a non-existing menu does not generate extra errors.

Description:

Dependencies:

1. USER INTERACTIVE
2. Test 7.2 is PASS

Test sequence:

1. Initialize:
    1. Clear DISPLAY
2. Execute “xdg-desktop-menu –uninstall –user SUB.menu”
    1. Expect exit 0
    2. (we permit warnings on STDERR or STDOUT)
3. Execute “xdg-desktop-menu –uninstall –user SUB.directory”
    1. Expect exit 0
    2. (we permit warnings on STDERR or STDOUT)
4. Ask user “Does menu SUB appear?”
    1. Expect NO

Notes/Questions:

Notes:

7. **Utility**: xdg-screensaver
    1. Generic tests.
    2. **Test**: activate
    Purpose: Verify screensaver activates immediately.
    Description:
    Dependencies:
    1. USER INTERACTIVE
    2. Running X server
    Test sequence:
    1. Instruct user "Do not touch machine until after screensaver activates"
    2. Execute "xdg-screensaver activate"
        1. Expect exit 0;
        2. expect no STDERR
        3. expect no STDOUT
    3. Ask user "Did screensaver activate immediately?"
        1. Expect YES
    Notes:
    3. **Test**: reset
    Purpose: Verify screensaver deactivates on command.
    Description:
    Dependencies:
    1. USER INTERACTIVE
    2. Running X server
    3. Test 9.2 (activate) is PASS
    Test sequence:
    1. Initialize
        1. Instruct user "do not touch the machine. The screen saver will turn on and off automatically."
        2. Execute "xdg-screensaver acitvate; sleep 1"
    2. Execute "xdg-screensaver reset"
        1. Expect exit 0
        2. expect no STDERR
        3. expect no STDOUT
    3. Ask user "Did screensaver deactivate automatically?"
        1. Expect YES
    Notes:
    4. **Test**: suspend delay – set shorter
    Purpose: Verify that setting the suspend delay works.
    Description:
    Dependencies:
    1. USER INTERACTIVE
    2. Running X server
    Test sequence:
    1. Initialize
        1. Ask user to verify that system is set to turn on the screen saver, and that the delay is > 1 minute.
    2. Instruct user "screensaver will activate after 1s – do not touch machine until it does."
    3. Execute "xdg-screensaver suspend 1s"

1. Expect exit 0
2. expect no STDOUT
3. expect no STDERR

4. Wait 1s
5. Ask user “Did screensaver activate?”
    1. Expect YES

Notes:

5. Test: suspend delay – set longer
Purpose: Verify that setting the suspend delay works.
Description:
Dependencies:
1. USER INTERACTIVE
2. Running X server

Test sequence:
1. Initialize
    1. Ask user to verify that system is set to turn on the screen saver, and set the delay to 2s
    2. Instruct user to verify screen saver activates after 2 s.
2. Instruct user “screensaver will activate after 10s – do not touch machine until it does.”
3. Execute “xdg-screensaver suspend 10s”
    1. Expect exit 0
    2. expect no STDERR
    3. expect no STDOUT
4. Ask user “How soon did the screen saver activate? (2 or 10)
    1. Expect 10

6. Test: restore
Purpose: Verify user's configuration is restored.
Description:
Dependencies:
1. USER INTERACTIVE
2. Running X server
3. Test 9.4/9.5 (suspend delay) are PASS
4. User configuration settings

Test sequence:
1. Initialize
    1. Ask user to disable screensaver.
    2. Execute “xdg-screensaver suspend 2s; sleep 1”
2. Instruct user “Do not touch the machine for 3 seconds.”
3. Execute “xdg-screensaver restore”
    1. Expect exit 0
    2. expect no STDOUT
    3. expect no STDERR
4. Ask user “Did screensaver activate?”
    1. Expect NO

Notes:

7. Test: status – basic no screensaver
Purpose: Verify we can get the status correctly.
Description:
Dependencies:

1. Running X server
2. User configuration settings

Test sequence:

1. Initialize
    1. ask user to disable screensaver
2. Execute “xdg-screensaver status”
    1. Expect exit 0
    2. Expect no STDERR
    3. Expect STDOUT to match “Enabled: false” exaclty.

Notes:

8. **Test**: status – start with no screensaver

Purpose: Verify we can get the status correctly.

Description:

Dependencies:

1. Running X server
2. User configuration settings

Test sequence:

1. Initialize
    1. ask user to disable screensaver
2. Execute “xdg-screensaver status”
    1. Expect exit 0
    2. Expect no STDERR
    3. Expect STDOUT to match “Enabled: false” exaclty.
3. Execute “xdg-screensaver suspend 1m”
4. Execute “xdg-screensaver status”
    1. Expect exit 0
    2. Expect no STDERR
    3. expect STDOUT to match “Enabled: true” exactly
5. Execute “xdg-screensaver reset”
6. Execute “xdg-screensaver status”
    1. Expect exit 0
    2. expect no STDERR
    3. expect STDOUT to match “Enabled: true” exaclty
7. Execute “xdg-screensaver restore”
8. execute “xdg-screensaver status”
    1. Expect exit 0
    2. expect no STDERR
    3. expect SDTOUT to match “Enabled: false” exactly

Notes:

9. **Test**: status – start with enabled screensaver

Purpose: Verify we can get the status correctly.

Description:

Dependencies:

1. Running X server
2. User configuration settings

Test sequence:

1. Initialize
    1. ask user to set screen saver to more than 5m delay.

2. Execute “xdg-screensaver status”
    1. Expect exit 0
    2. Expect no STDERR
    3. Expect STDOUT to match “Enabled: true” exaclty.
3. Execute “xdg-screensaver suspend 1m”
4. Execute “xdg-screensaver status”
    1. Expect exit 0
    2. Expect no STDERR
    3. expect STDOUT to match “Enabled: true” exactly
5. Execute “xdg-screensaver reset”
6. Execute “xdg-screensaver status”
    1. Expect exit 0
    2. expect no STDERR
    3. expect STDOUT to match “Enabled: true” exaclty
7. Execute “xdg-screensaver restore”
8. execute “xdg-screensaver status”
    1. Expect exit 0
    2. expect no STDERR
    3. expect SDTOUT to match “Enabled: true” exactly

Notes:

8. Utility: xdg-su
   1. **Test**: no argument test
      Purpose: Ensure that xdg-mime exits with an error and usage message when called without required arguments.
      Description: Run the utility with no arguments and check the output.
      Dependencies:
      Test sequence:
      1. Initialize:
         1. Clear DISPLAY
      2. Execute "xdg-mime"
         1. Expect nonzero exit code.
         2. Expect no STDOUT
         3. Expect STDERR

      Notes:
   2. **Test**: su test
      Purpose: Verify that the utility executes a command with an alternate user.
      Description: Execute a command to return the current user with xdg-su and see that it matches the test user.
      Dependencies:
      1. USER INTERACTIVE
      2. Running X server
      3. A test user account separate from the one the tests are running in
      4. whoami

      Test sequence:
      1. Execute "xdg-su -u testuser -c whoami
      2. Ask the user "does the password prompt dialog appear?"
         1. Expect YES
      3. Instruct the user "Input the password for testuser & click OK"
         1. Expect exit 0
         2. Expect STDOUT to match testuser exactly
         3. Expect no STDERR

      Notes:
   3. **Test**: no -c argument (bug 7136)
      Purpose: verify that xdg-su exits correctly when called without -c. (See bug 7136)
      Description: Run xdg-su without -c and expect an error/usage message.
      Dependencies:
      1. USER INTERACTIVE
      2. Running X server

      Test sequence:
      1. Execute "xdg-su whoami"
         1. Expect non-zero exit.
         2. Expect STDOUT
      2. Ask user "were you prompted for a password?"
         1. Expect NO

      Notes:

Use case test descriptions:

1. Associate an icon (xdg-icon) with a mime type (xdg-mime). Open a directory containing a file of that type (xdg-open) and verify that the icon displayed.
2. Associate a mime type with an application test to see that
    1. The application is returned by 'xdg-mime query default FILE'
    2. The application is started by 'xdg-open FILE'
3. RPM use case. Create an RPM to install a dummy application and mime type using xdg-utils. Verify files of that type are displayed correctly and the application opens that type.