

What changeForms is

ChangeForms is a tool that creates new PDF documents from PDF documents containing forms. The new documents can be “partially completed” with values extracted from a database, or they can be “totally completed and frozen”. The resulting documents are single PDF documents which can be stored and transferred with no dependencies on other files (in contrast to Acrobat FDF format where FDF files require a separate associated PDF file).

Interfaces

Change forms has command line interfaces as well as Web/CGI interfaces. It can also be used within other programs.

There are two modes for the command line interface. To dump an XML prototype description of the form within a document use the command line

```
changeforms.py --dumpxml fromdocument.pdf > xmlparameters.xml
```

To create a new document containing a modified form use the command line.

```
changeforms.py --rewrite fromdocument.pdf todocument.pdf parameters.xml
```

For more information on the XML input parameters please see the companion document “**How to integrate Changeforms into an Application Server**”.

The demo interfaces assumes a web server that supports CGI such as Apache and a Python installation on the web server machine with reportlab and the rlextra/pagecatcher and rlextra/radxml modules.

There are two included web interfaces for these components:

1. Change form database prototype. This provides a version of change forms which is integrated with an SQL database backend.
2. Change form CGI demo. This provides a simpler example usage for the changeforms functionality without database integration.

The functionality may also be used directly in other programs.

Database Prototype Installation

The prototype is designed with a Gadfly SQL database backend – it is expected that any standard SQL engine with a standard DBAPI python interface may be used in place of Gadfly with little effort.

1. Create the tables. The basic table definitions are provided in `tables.sql`. For non-Gadfly engines you may wish to change the definitions. In particular **make sure that there are no size restrictions or 8-bit data problems on the**

- pdfText** attributes because these attributes store the complete binary content of PDF files. In the case of Gadfly the tables and database can be created using the `initDB` function of the `databaseOps.py` python module. For other database engines use the SQL interface to execute the table definitions.
2. Configure the `databaseOps.py` python module for your engine. In particular if you are not using Gadfly you will need to edit the `connection()` function to connect to your database using the appropriate python interface, user, and password.
 3. Copy the `fillform.cgi` CGI script into a CGI directory. Edit the first line of the script to refer to the Python executable. Edit the other administration parameters for your server. Make sure that the CGI script and directory have the right file permissions to enable CGI execution. Make sure that the HTTP server recognizes the CGI directory as a directory that contains CGI executables. You may also need to create the directory for storing PDF files, making it writable and readable by the CGI user.
 4. At this point you should be able to access the `fillform.cgi` script from a browser (for example via <http://myserver/cgi-bin/fillform.cgi>).
 5. Use the `fillform.cgi` script to set up the database. First create a status; then upload a form; then associate the form with the status; then create an agent. At this point you should be able to fill in the form for the agent. The example form in `changeforms/projects/test/bridge.pdf` provides an appropriate test PDF form to upload.

Change form CGI demo installation

To install the CGI demo using a web server that supports CGI such as Apache:

1. Copy the cgi scripts `change.cgi` and `changeform.cgi` into a CGI directory.
2. Edit the ADMINISTRATION PARAMETERS in both of these scripts at the top. Also edit the first line of the scripts to locate the python executable correctly.
3. You may need to create a `pdfout` directory writable by CGI processes.
4. Note that in `change.cgi` the `pdfouthttp` parameter must be an **absolute URL** (not a relative URL) since the PDF form is cached locally (on MSIE) and loses the URL context in the process.
5. Copy the `changeforms` directory parallel to the CGI directory and make sure it is readable by CGI processes.
6. Set any permissions required for cgi scripts for `change.cgi` and `changeform.cgi` and test the scripts from the command line. `Changeform` should show a “choose projects” page and `change.cgi` should complain that a required parameter is missing. Other errors requi
7. Test the system by executing the `changeform.cgi` via the server script (usually at <http://server/cgi-bin/changeform.cgi>). It should just work.

Installing a new form in the demo

1. Design the form using Word or another design tool that supports printing (to PDF). Insert rectangles to contain every radio button. Leave spaces for text boxes and select boxes. Don't design the form to use check boxes or combo boxes as these are not supported yet.
2. Print the form design (with blanks for form elements) to PDF.
3. Edit the PDF design using Acrobat. Use the Acrobat Forms Tool (third from the bottom on the left menu bar in Acrobat 4 on Windows) to insert the form elements. Radio buttons should have no border or background and the "style" should be "solid".
4. Add a submit button to the form. The submit button must have a mouse down action of "submit form". The URL text can be anything (eg, <http://www.yahoo.com>) but the encoding for the URL submission must be URL encoding (not acrobat encoding).
5. If you want to put the new form in a new project create a new directory under changeforms/projects/NEWNAME.
6. Add the new form to the new project directory.
7. Test the new form with changeform.cgi (the new form should be automatically detected by the demo).

Miscellaneous Notes

If you get a "not indexable" error: Be sure the form contains a button named "submit" with down mouse action "submit form". The encoding for the submit action must be URL encoding, not acroform encoding.

Don't use check boxes in the form – use a radio button with two options or a selection box with two entries instead.

When a text box is marked "read only" by changeforms the border annotation is deleted – but other elements keep their border annotation even when the form has been frozen. If you don't want check boxes to have border annotations in the final frozen form make sure the source form check boxes don't have border annotations.

When a form is "frozen" the submit button and any other push button on the form are made invisible (since they have no purpose as read only elements).

The present use of Gadfly will slow down as the number of PDF files gets large. For better performance change the implementation to use Gadfly in server mode or port the application to another database engine.