AcroTeX Software Development Team

AcroTeX eDucation Bundle
The eq2db Package

D. P. Story

July 14, 2016

http://www.acrotex.net
dpstory@acrotex.net

Table of Contents

1 Introduction **3**
1.1 Introduction to HTML submission 3

2 The Distribution and Requirements **4**

3 The eqRecord option **4**
3.1 Field Values Processed by eqRecord.asp 6
- "Hard-wired" Fields 6
- Fields with Hierarchal Names 6

3.2 Mapping PDF Field Names onto DB Field Names 7
3.3 Adding more Hidden Fields 8
3.4 Setting Up 9
- Configuring the Server 10
- Set Up the Demo Files 10
- Comments on Demo Files 10

3.5 Accessing Results in the DB 11

4 The eqText Option **11**
4.1 Submit as FDF 11
- Field Values Processed by eqText.asp 13
- Setting up the Script 14

4.2 Submit as HTML 14

5 The eqEmail Option **15**
5.1 Submit as FDF 16
- Field Values processed by eqEmail.asp 17
- Setting up and Modifying the Script 17
- References 18

5.2 Submit as HTML 18

6 The tagged option **20**

7 The custom Option **22**

References **23**

1. Introduction

Short Summary: Converts a self-contained `exerquiz` `quiz` to one that is submitted to a server-side script. ◀

The `eq2eb` package is meant to be used with the `exerquiz` package, one of the components of the **AcroTEX eDucation Bundle** (`ctan.org/pkg/acrotex`). The package redefines the 'End Quiz' button (of the `quiz` environment) appropriately so that when the user clicks the 'End Quiz' control, the results of the quiz are sent to a server-side script. The `eq2db` Package has six options:

1. `eqRecord`: Save quiz data to a database, such as Microsoft Access. (Supported for FDF submission.)
2. `eqEmail`: Email quiz data to the instructor. (Supported for FDF and HTML submission.)
3. `eqText`: Save quiz data to a tab-delimited text file. Can then be opened and analyzed using Microsoft Excel. (Supported for FDF and HTML submission.)
4. `custom`: A hook for developers to use `eq2db` with their own script.
5. `tagged`: Write quiz data in an XML-like data stream.
6. `inclkey`, `exclkey` (new in v2.2) When `inclkey` is specified in the option list of `eq2db`, the <key> tag is included in the XML-structure when the `tagged` option is also specified; `inclkey` is the default and need not be specified. When `exclkey` is specified, then the <key> tag it not included in the XML-structure. See Section 6 for discussion of the <key> tag.
7. `submitAs=FDF|HTML|XML`: (new in v2.0) The `submitAs` option allows the document author to submit as FDF, HTML, or XML. The scripts provided by this package are designed FDF submittal and HTML submittal. Scripts for an XML submittal type are left as an exercise.

   With regard to HTML submission, read Section 1.1 below.

For FDF submittal, accompanying each of the first three options listed above is a server-side script. These are ASP pages written with VBScript. A Windows server running Microsoft IIS is required; Adobe *FDF Toolkit* is also required to be installed on the server.

1.1. Introduction to HTML submission

My upcoming book (if it ever comes), *AcroTEX eDucation System Tools: LATEX for interactive PDF documents* [1], has a rather extensive chapter titled "Server-side Scripting and the `eq2db` Package." In the course of writing that chapter, I decided to throw in the towel on the *FDF Toolkit*, the mainstay for FDF submission, and to look at HTML submission. Yes, PDF form data can be submitted as HTML form data. As result, I wrote a number of scripts for general purpose PDF form documents and for quiz documents (see Tables 3 and 4); identical functioning scripts were written for the Windows server-platform using ASP and VBScript and for the Linux server-platform using PHP.

AeB demonstration websites. These two websites were created to illustrate server-side concepts and examples brought up in the book. The links on these pages demonstrate the two sets of scripts listed in Tables 3 and 4, and much more.

`http://faculty.nwfsc.edu/web/math/storyd/dps/`

The above Web page is on a Windows server hosted by Northwest Florida State College (NWFSC). The server-side script is written as ASP pages using VBScript.

`http://www.acrotex.net/dps/`

This Web page is on a Linux server hosted by AcroTEX.Net. The server-side script is written with PHP.

A parallel development of all scripts is maintained for these two server platforms. You are invited to navigate your Web browser to each of these URLs and experience forms submission from a PDF document.

2. The Distribution and Requirements

The `eq2db` Package is distributed with the following files:

1. `eq2dbman.pdf`: This document, the documentation for the LaTeX package `eq2db` and its related files.
2. `eq2db.dtx` and `eq2db.ins`: The LaTeX package with its installation file.
3. `eqRecord.asp`: An ASP script for saving `exerquiz` quizzes to a database.
4. `eqEmail.asp`: An ASP script for forwarding `exerquiz` quizzes via e-mail.
5. `eqText.asp`: An ASP script for saving `exerquiz` quizzes to a tab-delimited file.
6. `quiz1.tex`, `quiz2.tex`, `quiz3.tex`, `quiz4.tex` (quizzes 1 and 2 demo use the `eqRecord` option; `quiz3` demos `eqEmail`, and `quiz4` demos `eqText`.)
7. `eqQuiz.mdb`: A Microsoft Access 2000 database is used with the demostration files `quiz1.tex` and `quiz2.tex`.

The `eq2db` package requires the `exerquiz` package to create online quizzes and to create supporting buttons and text fields. The `exerquiz` package, in turn, assumes a number of packages; a listing of these can be found in the documentation for the AcroTEX eDucation Bundle.

3. The `eqRecord` option

In this section we describe how to create an online quiz that is to be submitted to the server-side script `eqRecord.asp`[1]. For this option, quiz results are saved to a database. The `eqRecord` option and its corresponding script `eqRecord.asp` require that Adobe's FDF Toolkit is installed on your Windows server.

FDF Toolkit required

► `quiz1.tex` and `quiz2.tex` are the demo files for this option.

[1] The script `eqRecord.asp` comes with absolutely *no guarantees*. Extensive testing should be made on your own system to assure yourself script is reliable enough to use in practice. Feel free to modify the script to suite your needs.

In the preamble. The steps to create a quiz to be submitted to eqRecord are simple enough. First, the preamble of your document should look something like this:

```
\documentclass{article}
\usepackage[designi]{web}
\usepackage{exerquiz}
\usepackage[eqRecord,tagged]{eq2db}
```

You may have other packages loaded, as well as other options for web and exerquiz. The eq2db package was designed to be seamless in the following sense: If the eq2db package is not loaded, then an exerquiz quiz is self-contained, that is, it is not submitted to a server-side script; if eq2db package is loaded, then the quizzes are submitted to a server-side script. The tagged option may be optionally included in the option list of eq2db, shown in gray.

In the body. Next, you write your exerquiz quiz:

```
\eqSubmit{http://www.example.edu/scripts/eqRecord.asp\#FDF}
    {QuizIt}{Math101}
\begin{quiz*}{Quiz1}
Answer each of the following. Passing is 100\%.
\begin{questions}
\item ...
...
\item ...
\end{questions}
\end{quiz*}\quad\ScoreField\currQuiz\CorrButton\currQuiz
```
(1)

Preceding the quiz is the \eqSubmit command, defined in the exerquiz, it takes three parameters.

```
\eqSubmit{submitURL}{dbName}{dbTable}
```
(2)

Description of the parameters. The \eqSubmit command provides essential information use by the server-side scripts. The parameters of \eqSubmit are briefly described.

submitURL is the URL to the script eqRecord.asp#FDF. Internally, the value of this parameter is saved in the text macro \eq@CGI. Note that the name of the script eqRecord.asp has #FDF appended (the hash mark '#' followed by the fragment identifier FDF).

dbName is the name of the database to use. My examples use ODBC, with QuizIt as the DSN (data source name).

dbTable is the name of the table into which the data record is to be inserted (this is Math101 in the example above).

The three arguments of the command \eqSubmit are used to define the values of \db@Name, \db@Table and \eq@CGI, respectively, the values of which are used to populate the hidden fields.

Assuming eqRecord.asp is installed on your web server, and the *FDF Toolkit* has also been installed (see Section 3.4), we are ready to submit the quiz.

3.1. Field Values Processed by eqRecord.asp

The eqRecord script processes two classes of field data: certain "hard-wired" field data; and field data having a hierarchal name with a root of either ⟨*dbName*⟩ or IdInfo, e.g., Quiz1.numQuestions or IdInfo.Name.Last. Details of each of these follow.

- **"Hard-wired" Fields**

When you compile your source document using the package eq2db with the eqRecord option, the package creates three hidden text fields, with field titles of dbName, dbTable and quizName. These hidden fields are created under the 'End Quiz' button. At submit time, these fields are populated and submitted. Below is an enumerated list of these three fields with a brief description of each:

1. dbName: The name of the database to which the data is to be saved. (I've been using ODBC to reference the database.)
2. dbTable: The name of the table within dbName where the data is to be stored.
3. quizName: The name of the quiz or test. Name would uniquely characterize the quiz/test the student has taken.

The first two hidden fields are populated by the second and third arguments, respectively, of \eqSubmit, as defined in display (2). The third piece of data, quizName, is obtained from the first required argument of the quiz environment. See the skeleton example in display (1), there you'll see \begin{quiz*}{Math101}. The argument 'Math101' is the quiz name and is passed to the server-side script as the value of the quizName hidden text field.

- **Fields with Hierarchal Names**

Other than the fields described in "Hard-wired" Fields, eqRecord.asp processes only fields with hierarchal names that have a root name of ⟨*quizName*⟩ (see "Hard-wired" Fields on page 6) or a root name of IdInfo.

quizName (\currQuiz): quizName is the field title of one of the hidden fields, its value is picked up as the first argument of the quiz environment; the value is stored in the text macro \currQuiz.

In addition to the "hard-wired" hidden fields described earlier, there are actually three more hidden fields (under the 'End Quiz' button) with root name ⟨*quizName*⟩. These are

- \currQuiz.numQuestions: The number of questions in the quiz
- \currQuiz.numCorrect: The number of correct questions
- \currQuiz.Responses: a list of all the responses of the user

The values of these fields are also sent to eqRecord.asp. There is a mechanism for creating more hidden fields the values of which are sent to the script. The technique for doing this will be discussed later.

IdInfo: As mentioned earlier, eqRecord.asp processes fields that use a hierarchal naming convention, with root name of ⟨*quizName*⟩ or IdInfo. Fields whose root name is IdInfo are meant to hold information about the person taking the quiz: first name, last name, student number, etc.

For example, you can create fields the user fills in for self-identification. In the preamble, you can define new commands:

```
% User's First Name
\newcommand\FirstName[2]{\textField{IdInfo.Name.First}{#1}{#2}}

% User's Last Name
\newcommand\LastName[2]{\textField{IdInfo.Name.Last}{#1}{#2}}

% User's SSN
\newcommand\SSN[2]{\textField[\MaxLen{11}
    \AA{\AAKeystroke{AFSpecial_Keystroke(3);}
        \AAFormat{AFSpecial_Format(3);}
    }]{IdInfo.SSN}{#1}{#2}}
```

Note that I've defined these fields so that their names follow a hierarchal name structure, with a root of IdInfo. The two required arguments are the dimensions of the text field being constructed. For example, \FirstName{100bp}{10bp} creates a text field with a title of IdInfo.Name.First which is 100bp wide and 10bp high.

► See the demo file quiz1.tex for examples.

Additional IdInfo fields can be constructed.

3.2. Mapping PDF Field Names onto DB Field Names

eqRecord.asp is a quasi-general script for mapping the values of PDF fields into corresponding fields in a database. The way eqRecord is set up, the DB field name is derived from the PDF field name. For example, the value of the PDF field Quiz1.numQuestions is stored in the DB under the DB field name of numQuestions.

For field names such as IdInfo.Name.Last, we can't map this into the DB field name Name.Last because for some databases (notably, Microsoft Access) database field names containing a 'dot' are not legal. As a work around, eqRecord.asp strips all dots from the field name, and replaces them with the value of the VB Script variable dotReplace. The definition of dotReplace in eqRecord.asp is

```
Dim dotReplace : dotReplace = "_"
```

That is, a 'dot'(.) is replaced by an 'underscore' (_).

The table below illustrates the mapping of PDF field name onto database field names. Suppose the quizName is Quiz1:

PDF Field Name	DB Field Name	Required
quizName (e.g., Quiz1)	quizName	Yes
Quiz1.numQuestions	numQuestions	Yes

PDF Field Name	DB Field Name	Required
Quiz1.numCorrect	numCorrect	Yes
Quiz1.Responses	Responses	Yes
IdInfo.Name.Last	Name_Last	No
IdInfo.Name.First	Name_First	No
IdInfo.SSN	SSN	No
N/A	TimeOfQuiz	Yes

The last entry needs comment: eqRecord.asp generates a time stamp when a quiz is processed. This time stamp is stored in a field with a name of TimeOfQuiz. This is a required field in your database.

3.3. Adding more Hidden Fields

When you add fields with a hierarchal name with a root of \currQuiz or IdInfo—whether hidden or not—the values of these fields will be submitted and processed by eqRecord.asp. When you add fields in your document, there should be a corresponding DB field to receive this data (see Section 3.2 for naming conventions). You can also add hidden fields to transmit information about the quiz that the user does not need to see.

Example. Suppose you have a point value to each question and want to report the point score (rather than the number missed). In this case, you would use the \PointsField field instead of the \ScoreField (though you could use both).

\PointsField has a hierarchal name, but its root does not begin with \currQuiz; consequently, the value of this field is not submitted to eqRecord.asp. (The value of this field is a string what is meant to be read by the user; it reads, for example, "Score: 16 out of 20"; we don't want this string submitted anyway, we would want the point score (16) and the total points (20) submitted.) We want to transmit the points scored and the total number of points to the server-side script.

The eq2db package defines two helper commands for creating hidden fields (these hidden fields are hidden under the 'End Quiz' button); these are \addHiddenField and \populateHiddenField.

Suppose we had a quiz, Quiz1, in which we wanted to report points scored and total points. First, we need to add two hidden fields, we'll call them Quiz1.ptScore and Quiz1.nPointTotal; we create the hidden fields using \addHiddenField:

```
\addHiddenTextField{Quiz1.ptScore}{}
\addHiddenTextField{Quiz1.nPointTotal}{\theeqpointvalue}
```

The first parameter is the title (name) of the field, the second parameter is the default value. For the first line, no default value is given; for the second line, a default value of \theeqpointvalue is given. The counter \eqpointvalue contains the total number of points for the quiz so we can insert that value at latex compile-time.

The value of the first field added, Quiz1.ptScore, is not known until a user takes a quiz and submits results. The eq2db command \populateHiddenField inserts the necessary JavaScript that would populate the specified field. For example,

```
\populateHiddenField{Quiz1.ptScore}{ptScore}
```

The first parameter is the field name, the second is the value the field is to hold when the user clicks on 'End Quiz'. The command

```
\populateHiddenField{fieldname}{fieldvalue}
```

expands to,

```
this.getField("fieldname").value = fieldvalue;
```

which is the JavaScript for populating the field, is inserted into the code just prior to the submission of the data.

► See second quiz in the sample file `quiz1.tex` for a complete example of creating a quiz with weight set for each question, and for reporting the point score and total points. ◄

The table below lists some variables (both LaTeX and JavaScript) that might be useful in extracting desired information from quiz for submittal.

Variable	Description
`Score`	Score of the user with one point per problem. This value is always reported in the hidden text field `quizName.numCorrect`. This is a JavaScript variable known when the user finishes the quiz. This value is always reported in `quizName.numQuestions`, the hidden text field.
`\thequestionno`	The number of questions in the current quiz. A LaTeX macro; known at compile-time.
`Responses`	A comma-delimited list of all responses of the user. A JavaScript variable known when the user finishes the quiz. Always reported in the hidden field `quizName.Responses`
`ptScore`	The point score of the user's quiz. A JavaScript variable, known when the user finishes quiz.
`\theeqpointvalue`	The total number of points in the current quiz. A LaTeX macro, known at compile-time.
`pcScore`	The score expressed as a percent. A JavaScript variable known when the user finishes the quiz.
`quizGrade`	The letter grade of the user's quiz. A JavaScript variable, known at "run-time".
`\eqGradeScale`	The grade scale for the quiz. A LaTeX macro, known at "compile-time".

3.4. Setting Up

In this section, we briefly discuss configuring your server and setting up the demo files that accompany this distribution.

- **Configuring the Server**

On the server side, in order for eqRecord.asp to run correctly, Microsoft Internet Information Server (IIS), version 4.0 or greater, is needed. The script eqRecord.asp needs to be placed where ASP scripts have execute permissions.

The eqRecord.asp uses the *Adobe FDF Toolkit*[2], version 6.0. Follow the directions for installation contained in the accompanying documentation.

Install eqRecord.asp in a folder (perhaps called Scripts) designated to execute scripts. If you don't have such a folder, then the following steps explain how to create a virtual directory through IIS that points to this folder.

1. Create a new folder on the system (Scripts, for example). Its recommended location is inside the Inetpub folder.
2. Place eqRecord.asp in this newly created folder.
3. In the MMC snap-in for IIS, create a virtual directory by right-clicking on the Default Web Site and selecting New > Virtual Directory.
4. Type "Scripts" (or whatever the name of the folder you created in Step 1) as the alias for the virtual directory, and then link it to the physical directory you created in Step 1.
5. Make sure that "Script execution" privileges are enabled. If not, enable them.

- **Set Up the Demo Files**

Place the Access 2000 database, QuizIt.mdb, in a folder that is accessible by the server, such as the root level of your web server, or perhaps, a folder dedicated to database files. Register the database with the ODBC Data Source Administrator as a System DSN under the System Data Source Name of "QuizIt".

Modify the first parameter of \eqSubmit in quiz1.tex and compile (latex), then create quiz1.pdf using Acrobat Distiller, pdftex or dvipdfm, and place it on your server. Test by opening quiz1.pdf in your web browser and taking a sample test. Good luck, I hope it works!

- **Comments on Demo Files**

quiz1.tex: This file has two quizzes, Quiz1 and Quiz2. Quiz1 has no frills, the user enters his/her name and SSN, takes the quiz, and results are saved to the eqQuiz.mdb Access database. Quiz2 is a bit more interesting as it demonstrates how to assign points to each question, and how to report the results. The techniques used in this quiz were discussed in Section 3.3.

quiz2.tex: The quizzes of quiz1.tex did not check whether the user is enrolled in the class, and as such, is allowed to take the quiz. The demo file quiz2.tex demonstrates how to…

1. check whether the user has entered a valid SSN, that is, the SSN of someone enrolled in the class, or is permitted to take the quiz

[2] Currently located at the Acrobat Family Developer Center.

2. set a deadline date to take the quiz

3. set a time limit to take the quiz

Other variations are possible.

3.5. Accessing Results in the DB

The database that is to hold quiz results is placed on a web-server and consequently, is not easily accessible by the instructor. The ASP script GenericDB[3] developed by Eli Robillard, is a tool that can be used to access the database through your web browser. Using GenericDB, you can view, edit, update, add new and delete database records—and it's free! Check it out!

4. The eqText Option

In this section, we describe how to submit your data to a server-side script that save the data to a tab-delimited text field. The submitAs option can be used with the eqText option. With submitAs=FDF, the form data is submitted in the FDF format, the native form data format for PDF, the target script should be eqText.asp. When submitAs=HTML is used, the form data is submitted as ordinary HTML form data, which can then be processed by any public domain script; alternately, you can use one of the scripts in Table 3. These four scripts are available on the CD-ROM that accompanies the book *AcroTEX eDucation System Tools: LaTeX for interactive PDF documents* [1].

Option and return type	Script Files for Windows & Linux
eqText, HTML return	eqText_h.asp eqText_h.php
eqText, FDF return	eqText_hf.asp#FDF eqText_hf.php#FDF

Table 3: Scripts for eqText & submitAs=HTML options

4.1. Submit as FDF

FDF Toolkit required

We describe how to create an online quiz that is to be submitted to the server-side script eqText.asp[4]. With this option, you save your quiz data to a tab-delimited file. The *FDF Toolkit* is required to be installed on the Windows server.

► The demo file for this option is quiz4.tex.

[3] http://www.genericdb.com/

[4] The script eqText.asp comes with absolutely *no guarantees*. Extensive testing should be made on your own system to assure yourself script is reliable enough to use in practice.

In the preamble. Begin the document with the usual preamble for an exerquiz document, but include the eq2db package with the eqRecord option.

```
\documentclass{article}
\usepackage{amsmath}
\usepackage[⟨driver⟩,designi]{web} % driver: dvips, pdftex, xetex
\usepackage{exerquiz}
\usepackage[submitAs=FDF,eqText]{eq2db}
```
(3)

Shown in gray are the ⟨*driver*⟩ and submitAs options. The pdflatex and xelatex applications are automatically detected, so only dvips needs to be specified if you are building your PDF with Acrobat Distiller. The submission type is FDF by default, so submitAs=FDF need not be listed in the options of eq2db.

In the body. The following is a rough outline of an exerquiz quiz document that submits to eqRecord.asp, the preamble declarations of display (3).

```
\eqSubmit{http://www.example.edu/scripts/eqText.asp\#FDF}
    {c:/Inetpub/Data/math101.txt}{Math101}
\begin{quiz*}{Quiz1} Answer each of the following. Passing
is 100\%.
\begin{questions}
\item ...
...
\item ...
\end{questions}
\end{quiz*}\quad\ScoreField\currQuiz\CorrButton\currQuiz
```
(4)

Notice that '#FDF' is appended to the path, this informs the PDF viewer to expect the server-side script to return FDF form data.

Preceding the quiz is the \eqSubmit command, defined in the exerquiz, it takes three parameters.

```
\eqSubmit{submitURL}{pathToTabFile}{courseName}
```
(5)

Description of the parameters. The \eqSubmit command provides essential information use by the server-side scripts. Its parameters are briefly described.

submitURL is the URL to the server-side script eqText.asp. Internally, the value of this parameter is saved in the text macro \eq@CGI.

pathToTabFile is the path to the tab-delimited file on the server where the script is to save the data. On Windows, *pathToTabFile* is the physical (absolute) path to the file, on Linux it can be a relative path (relative to the folder containing the executing script). Internally, this value is saved in the text macro \db@Name.[5]

courseName is the course name of the class taking the quiz. Internally, this value is saved in the text macro \db@Table.[5]

[5] Choice for the names of \db@Name and \db@Table were originally based on the eqRecord option where data is saved to a database.

- **Field Values Processed by eqText.asp**

The eqText script processes two classes of field data: certain "hard-wired" field data; and field data having a hierarchal name with a root of either ⟨*dbName*⟩ or IdInfo, e.g., Quiz1.numQuestions or IdInfo.Name.Last. Details of each of these follow.

"Hard-wired" Fields. When you compile your source document with the package eq2db with the eqText option, the package creates a number of hidden text fields. These hidden fields are created under the 'End Quiz' button. At submit time, these fields are populated and submitted along with the quiz data. Below is an enumerated list of these hidden fields with a brief description of each:

1. pathToTxtFile: The value of second argument of \eqSubmit populates this text field.
2. courseName: The value of the third parameter of \eqSubmit populates this text field.
3. quizName: The name of the quiz or test, this is \currQuiz. Name should uniquely characterize the quiz/test the student is taking.

The first two hidden fields are populated by the second and third arguments, respectively, of \eqSubmit, as defined in display (5). The third piece of data, quizName, is obtained from the first required argument of the quiz environment. See the skeleton example in display (4), there you'll see \begin{quiz*}{Math101}. The argument 'Math101' is the quiz name and is passed to the server-side script as the value of the quizName hidden text field.

Fields with Hierarchal Names. Other than the fields described in "Hard-wired" Fields, eqText.asp processes only fields with hierarchal names that have \currQuiz as a root name (see "Hard-wired" Fields above) or a root name of IdInfo.

quizName (\currQuiz): quizName is the field title of one of the hidden fields, its value is picked up as the first argument of the quiz environment; the value is stored in the text macro \currQuiz.

In addition to the "hard-wired" hidden fields described earlier, there are actually five more hidden fields (under the End Quiz button) with root name \currQuiz. These are,

- \currQuiz.numQuestions: The number of questions in the quiz.
- \currQuiz.numCorrect: The number of correct questions.
- \currQuiz.Responses: A list of all the responses of the user.
- \currQuiz.ptScore: The point score the student received on completion of the quiz.
- \currQuiz.nPointTotal: The total number of points in the quiz.

The values of these fields are also sent to the server-side script for the eqText option. There is a mechanism for creating more hidden fields the values of which are sent to the script. The technique for doing this is discussed in 'Adding more Hidden Fields' on page 8.

IdInfo: As mentioned earlier, the eqText scripts process fields that use a hierarchal naming convention, with root name of \currQuiz or IdInfo. Fields whose root name is IdInfo are meant to hold information about the person taking the quiz: first name, last name, student number, and so on. Refer to the paragraph titled **IdInfo** on page 7 for more information.

- **Setting up the Script**

On the server side, in order for eqText.asp to run correctly, Microsoft Internet Information Server (IIS), version 4.0 or greater, is needed. The script eqText.asp should be placed where ASP scripts have execute permissions.

4.2. Submit as HTML

When the eqText and submitAs=HTML options are specified for eq2db, the form data are submitted as ordinary HTML form data. For the eqText option, it is meant that the server-side script save the form data as a tab-delimited file. You can develop your own script for doing this, or you can use one of the scripts of Table 3, available on the CD-ROM of the book *AcroTEX eDucation System Tools: LaTeX for interactive PDF documents* [1]. The advantage of submitting as HTML is that the *FDF Toolkit* is not needed, and the techniques for writing a server-side script are well known.

The preamble of your document should look something like the following.

```
\documentclass{article}
\usepackage[designi]{web}
\usepackage{exerquiz}
\usepackage[eqText,submatAs=html,tagged]{eq2db}
```

The tagged option is recommended for the eqText option but no required. You may have other packages loaded, as well as other options for web and exerquiz. The eq2db package was designed to be seamless in the following sense: If the eq2db package is not loaded, then an exerquiz quiz is self-contained, that is, it is not submitted to a server-side script; if eq2db package is loaded, then the quizzes are submitted to a server-side script.

In addition to the introduction of the eq2db package in the preamble, there are two more important commands:

```
\eqSubmit{submitURL}{pathToTabFile}{courseName}
\rtnURL{returnURL}
```
(6)

\eqSubmit, defined in exerquiz, is required; \rtnURL, defined in eq2db, is required only when the return from the server is an HTML page, but is recommended.

Description of the parameters. The \eqSubmit command provides essential information use by the server-side scripts, \rtnURL provides a return URL that points to where the student should navigate after the quiz. Their parameters, as indicated in display (6), are briefly described.

submitURL is the URL of one of the server-side scripts in Table 3. Internally, the value of this parameter is saved in the text macro \eq@CGI.

pathToTabFile is the path to the tab-delimited file on the server where the script is to save the data. On Windows, *pathToTabFile* is the physical (absolute) path to the file, on Linux it can be a relative path (relative to the folder containing the executing script). Internally, this value is saved in the text macro \db@Name.[5]

courseName is the course name of the class taking the quiz. Internally, this value is saved in the text macro \db@Table.[5]

returnURL is the URL of the page the student should return to after finishing the quiz. The value of \rtnURL is required when the script returns an HTML page, and is useful when the return is FDF. Internally, \rtnURL defines the text macro \thisRtnURL to hold *returnURL*.

The scripts. The *submitURL* points to the server-side script that is to process the form data. The scripts of Table 3 were written for the upcoming book *AcroTEX eDucation System Tools: LaTeX for interactive PDF documents* and are available on the accompanying CD-ROM. Refer the book for the details of implementing these scripts.

HTML versus FDF return. The scripts of Table 3 are of two types: HTML return and FDF return. In the first case, after the student has submitted his/her quiz, the server-side script returns an HTML page to the browser window with an assessment of the student's efforts on the quiz. In the second case, the script sends back an alert dialog box message saying the data is received and saved; the quiz is self-marking for the student to review. Details are laid out in *AcroTEX eDucation System Tools: LaTeX for interactive PDF documents* [1].

5. The eqEmail Option

In this section, we describe how to submit your data to a server-side script that in turn sends an email message to a list of recipients, the body of the messages contains the student's quiz results. The submitAs option can be used with the eqEmail option. With submitAs=FDF, the form data is submitted in the FDF format, the native form data format for PDF, the target script should be eqEmail.asp. When submitAs=HTML is used, the form data is submitted as ordinary HTML form data, which can then be processed by any public domain script; alternately, you can use one of the scripts in Table 4. These four scripts are available on the CD-ROM that accompanies the book *AcroTEX eDucation System Tools: LaTeX for interactive PDF documents* [1].

eqEmail, HTML return	eqEmail_h.asp eqEmail_h.php
eqEmail, FDF return	eqEmail_hf.asp#FDF eqEmail_hf.php#FDF

Table 4: Scripts for eqEmail & submitAs=HTML options

5.1. Submit as FDF

FDF Toolkit required

We describe how to create an online quiz that is to be submitted to the server-side script eqEmail.asp[6]. For this option, quiz results are e-mailed to the instructor. The *FDF Toolkit* is required to be installed on the Windows server.

► quiz3.tex is the demo file for this option.

The steps to create a quiz to be submitted to eqEmail are simple enough. First, the preamble of your document should look something like this:

```
\documentclass{article}
\usepackage[designi]{web}
\usepackage{exerquiz}
\usepackage[eqEmail]{eq2db}
```

You may have other packages loaded, as well as other options for web and exerquiz. The eq2db package was designed to be seamless in the following sense: If the eq2db package is not loaded, then an exerquiz quiz is self-contained, that is, it is not submitted to a server-side script; if eq2db package is loaded, then the quizzes are submitted to a server-side script.

Next, you write your exerquiz quiz:

```
\eqSubmit{http://www.example.edu/scripts/eqEmail.asp\#FDF}
    {dpspeaker@example.edu}{Math101}
\begin{quiz*}{Quiz1} Answer each of the following. Passing is 100\%.
\begin{questions}
\item ...
...
\item ...
\end{questions}
\end{quiz*}\quad\ScoreField\currQuiz\CorrButton\currQuiz
```

Preceding the quiz is the \eqSubmit command, defined in the exerquiz.

```
\eqSubmit{submitURL}{recipients}{courseName}
```

The first parameter is the URL to the server-side script, in the example above, we have http://www.example.edu/scripts/eqEmail.asp#FDF (note the suffix of #FDF); the second parameter, *recipients*, is the e-mail address of the person to receive the quiz results; the third parameter, *courseName* is the course name. In the above example, the course name is Math101.

► If metarecipients is a comma-delimited list of e-mail addresses, then eqEmail.asp will parse the list, and use the first e-mail in the list in the From field of the e-mail. Other email addresses in the list also receive copies of the quiz results. ◄

[6]The script eqEmail.asp comes with absolutely *no guarantees*. Extensive testing should be made on your own system to assure yourself script is reliable enough to use in practice. Feel free to modify the script to suite your needs. If your improvements are noteworthy, please allow me to incorporate them into the basic eqEmail.asp for others to use.

There is another parameter of importance. The first argument of the quiz environment is the quizName of the quiz, this is Quiz1 in the above example.

Assuming eqEmail.asp is installed on your web server, and the *FDFToolkit* has also been installed (see Section 3.4), we are ready to submit the quiz. Below is the body of one of my test e-mails I made using quiz3.pdf:

```
Course Information
    Course Name: Math101
    Quiz: Quiz1
    TimeOfQuiz: 8/23/2003 8:12:40 PM

Student Results
    Name_First: Don
    Name_Last: Story
    SSN: 121212121
    Responses: Leibniz,2x e^(x^2),-1,-cos(x)
    numCorrect: 4
    numQuestions: 4
```

If you add more fields, as described in Section 3.3, this information should be appended to the body of the e-mail.

- **Field Values processed by eqEmail.asp**

Actually, eqEmail.asp is a variation of eqRecord.asp and many of the details of this section are the same Section 3.1.

The "hard-wired" fields are mailTo, courseName and quizName. These fields are hidden under the 'End Quiz' button, and are populated at submit time. As in the eqRecord option, the value of the quizName field comes from the argument of the quiz environment:

```
\eqSubmit{http://www.example.edu/scripts/eqEmail.asp\#FDF}
    {dpspeaker@example.edu}{Math101}
\begin{quiz*}{Quiz1} Answer each of the following. Passing is 100\%.
\begin{questions}
\item ...
...
\item ...
\end{questions}
\end{quiz*}\quad\ScoreField\currQuiz\CorrButton\currQuiz
```

In this quiz, the field with name quizName will have a value of Quiz1.

See paragraph "Fields with Hierarchal Names", page 6, as well as Section 3.3 on page 8 for the eqRecord option, the details are the same.

- **Setting up and Modifying the Script**

On the server side, in order for eqEmail.asp to run correctly, Microsoft Internet Information Server (IIS), version 4.0 or greater, is needed. The script eqEmail.asp should be placed where ASP scripts have execute permissions. There are two methods of sending e-mail:

1. CDONTS: This method (which is commented out by default) can be used on an NT server. Uncomment if you want to use CDONTS, and comment out the CDOSYS code lines that follow.

2. CDOSYS: This can be run on a Win2000 or WinXP server.

The script needs to be modified appropriate to your server, in particular, search down in eqEmail.asp for the configuration line

```
eqMail.Configuration.Fields.Item
    ("http://schemas.microsoft.com/cdo/configuration/smtpserver")
        = "mySMTP"
```

replace mySMTP with your SMTP server.

The subject heading of the returning e-mail has the following format:

```
Quiz Results: Quiz1 of Math101
```

If you want another subject format, modify eqMail.Subject as desired.

- **References**

The following links were used as a reference in the development of the Email.asp script.

- CDOSYS:
  - Invision Portal Tutorial: CDOSYS email tutorial
  - MSDN: CDO for Windows 2000. The IMessage Interface. (Use MIE to view this page.)
  - ASP 101 Sending Email Via an External SMTP Server Using CDO
- CDONTS
  - Juicy Studio The ASP CDONTS Component
  - DevASP Sending Mail from ASP with CDONTS.NewMail Object

5.2. Submit as HTML

When the eqEmail and submitAs=HTML options are specified for eq2db, the form data are submitted as ordinary HTML form data. For the eqEmail option, it is meant that the server-side script should take the incoming form data, build an email the body of which reports the quiz results, and send the message to a list of recipients. You can develop your own script for doing this or you can use the scripts provide with the book *AcroTEX eDucation System Tools: LaTeX for interactive PDF documents* [1]. The advantage of submitting as HTML is that the *FDF Toolkit* is *not needed*, and the techniques for writing a server-side script are well known.

The preamble of your document should look something like the following.

```
\documentclass{article}
\usepackage[designi]{web}
\usepackage{exerquiz}
\usepackage[eqEmail,submatAs=html]{eq2db}
```

The tagged option is *not recommended* for the eqEmail. You may have other packages loaded, as well as other options for web and exerquiz. The eq2db package was designed to be seamless in the following sense: If the eq2db package is not loaded, then an exerquiz quiz is self-contained, that is, it is not submitted to a server-side script; if eq2db package is loaded, then the quizzes are submitted to a server-side script.

In addition to the introduction of the eq2db package in the preamble, there are two more important commands:

```
\eqSubmit{submitURL}{recipients}{courseName}
\rtnURL{returnURL}
```
(7)

\eqSubmit, defined in exerquiz, is required; \rtnURL, defined in eq2db, is required only when the return from the server is an HTML page, but is recommended.

Description of the parameters. The \eqSubmit command provides essential information use by the server-side scripts, \rtnURL provides a return URL that points to where the student should go to next after the quiz. Their parameters, as indicated in display (7), are briefly described.

submitURL is the URL of one of the server-side scripts in Table 3. Internally, the value of this parameter is saved in the text macro \eq@CGI.

recipients is a comma-delimited list of email addresses to whom the quiz results are sent. For example,

```
dpspeaker@talking.edu,taofdps@talking.edu
```

The first email address is picked off and used in the 'From' field of the email while the whole list is entered into the 'To' field. Internally, this value is saved in the text macro \db@Name.[5]

courseName is the course name of the class taking the quiz. Internally, this value is saved in the text macro \db@Table.[5]

returnURL is the URL of the page the student should return to after finishing the quiz. The value of \rtnURL is required when the script returns an HTML page, and is useful when the return is FDF. Internally, \rtnURL defines the text macro \thisRtnURL to hold *returnURL*.

The scripts. The *submitURL* points to the server-side script that is to process the form data. The scripts of Table 4 were written for the upcoming book *AcroTEX eDucation System Tools: LaTeX for interactive PDF documents* [1] and are available on the accompanying CD-ROM. Refer the book for the details of implementing these scripts.

HTML versus FDF return. The scripts of Table 4 are of two types: HTML return and FDF return. From the student's point of view, his/her experience is the same as the `eqText` option with `submitAs=HTML`. In the first case, after the student has submitted his/her quiz, the server-side script, as sending out an email response to the ⟨*recipients*⟩, returns an HTML page to the browser window with an assessment of the student's efforts on the quiz. In the second case, the script sends back an alert dialog box message saying the data is received and results are sent to the instructor (the ⟨*recipients*⟩); the quiz is self-marking for the student to review. Details are presented in *AcroTEX eDucation System Tools: LaTeX for interactive PDF documents* [1].

6. The tagged option

When the student submits online quiz data to one of the server-side scripts, just prior to submission, the hidden field `\currQuiz.Responses` is populated with the student's answers. For example,

```
Newton,2x,-1,-cos(x)
```

is how the responses field is populated, by default. This data gives no indication of which are correct, how many points for each question, and so on.

To obtain more information about the quiz taken by the student and his responses, you can specify the `tagged` option. When this option is taken, the quiz data is written as a "XML-like" string. Figure 1 is an example of the data that populates the `\currQuiz.Responses` field and which is sent to the server when the `tagged` option is specified.

Referring to Figure 1, the XML string packs more information in it than the default string. The quiz name is `Quiz1`, the PDF file the student used is `quiz1.pdf` and there are four questions (`n="4"`). The first question was a text fill-in type, worth 3 points, the student responded correctly. The second question was a math fill-in worth 4 points, he missed this one with an answer of `2x`.

The root element `results` contains child elements or nodes called `question`. Each `question` node contains information about one of the questions in the quiz. The `question` node has several attributes, these attributes are seen in Figure 1 above:

- `n` is the question number. This is not the question number the user sees, but the value of an internal counter that increments each time a new question is posed.
- `type` is the type of question, possible values are `"text"` (text response), `"math"` (math response), `"mc"` (multiple choice), `"ms"` (multiple selection), `"grp"` (grouped response), and `"essay"` (multi-line text response). The latter is not graded by `exerquiz` but simply transmitted to the server.
- `ptype` is the problem (question) type, its value is normally `"na"` for 'not applicable'. This attribute is used to describe the question in more detail within the XML. The value of `ptype` is set within the LaTeX source file with `\QT{`*ptype*`}` command. Placement of this command is just after the `\item` command within the `questions` environment.

```
\item\PTs{4}\QT{limits} Compute $\lim_{n\to\infty}...$
```

```
<results id="Quiz1" file="quiz1.pdf" n=4>
    <question n=1 type="text" ptype="na" points="3" credit="3" correct="1">
        <value>Newton</value>
        <key>
             [&quot;Isaac Newton&quot;,&quot;Newton&quot;,
              &quot;I. Newton&quot;,&quot;Gottfried Leibniz&quot;,
              &quot;Leibniz&quot;]
        </key>
    </question>
    <question n=2 type="math" ptype="na" points="4" credit="0" correct="0">
        <value>2x</value>
        <key>2xe^(x^2)</key>
    </question>
    <question n=3 type="math" ptype="na" points="3" credit="0" correct="0">
        <value>-1</value>
        <key>2</key>
    </question>
    <question n=4 type="math" ptype="na" points="5" credit="5" correct="1">
        <value>-cos(x)</value>
        <key>-cos(x)</key>
    </question>
</results>
```

Figure 1: Example of a tagged XML string

Now, the `ptype` attribute will read `ptype="limits"`. If each problem is so tagged, the `ptype` attribute can be queried as part of larger quiz management system to view a student's effort with respect to all questions having the specified `ptype`. Such a system was never implemented, but the attribute is available.

- `points` is the number of points assigned to this question.
- `credit` is the amount of partial credit the student earned for this question. Questions types where this attribute can be nonzero are `"mc"`, `"ms"`, `"text"` (in response to a `\RespBoxTxtPC` question), and `"grp"`.
- `correct` is normally `"0"` (a wrong answer) or `"1"` (a correct answer); multiple selection (`"ms"`) and grouped (`"grp"`) questions use an array of values for the `correct` attribute, however. When calculating the score field (`\ScoreField`), the problem is either right or wrong; however, the calculation of the points field (`\PointsField`) takes into account the partial credit so a problem may be partially correct.

In Figure 1, the `<value>` tag represents the student's response to the question. The `<key>` tag, shown in gray, represents the author's answer to the question. The `<key>` tag may or may not be present depending on the options `inclkey` and `exclkey`. For a more detailed analysis of the XML-structure, study the document `taggedxml.pdf`, available on the CD-ROM that accompanies *AcroTEX eDucation System Tools: LATEX for interactive PDF documents* [1].

Reconstructing the quiz. Using the XML data as well as the information provided by the `IdInfo` collection of fields, it is possible to reproduce the PDF document exactly as the student submitted it. A demonstration file was developed for exactly this purpose and is available on the CD-ROM that accompanies *AcroTEX eDucation System Tools: LATEX for interactive PDF documents* [1]. The Acrobat application is required.

7. The `custom` Option

This option allows a script developer to utilize the macros of the `eq2db` Package to prepare an AcroTEX document to submit to a custom script. Simple create a file named `eq2dbcus.def` and include any custom creation of fields you may need for your script. When you then take the `custom` option,

```
\documentclass{article}
\usepackage[designi]{web}
\usepackage{exerquiz}
\usepackage[custom]{eq2db}
```

After the command definitions of `eq2db` are read, `eq2dbcus.def` is input. The rest is up to you.

► That's all for now! Hope you find the package useful. Now I really must get back my retirement. ☺

References

[1] *AcroTEX eDucation System Tools: LATEX for interactive PDF documents*, D. P. Story, in preparation. See pages 3, 11, 14, 15, 18, 19, 20, 21, and 22.