

# Self-Marking Quiz

Written for FEB 8.6

## Table of Contents

Design.....	1
Adapting to your Use Case.....	4
Need Help?.....	5

**DEMO - Quiz built using FEB**

**Question 1**

\* Schedule Adherence means

- Metric used in the call center to determine whether or not call center agents are working the amount of time they are scheduled to work.
- The degree to which agents stick to their schedulesSpecify Value 2
- None of the above
- Both 1 and 2

**Question 2**

What is VDN?

- Vector Directory Number
- Volunteer Driver NetworkSpecify Value 2
- Vector Defined Network
- Virtual Dialed NumberSpecify Value 4

**Question 3**

What is ACD?

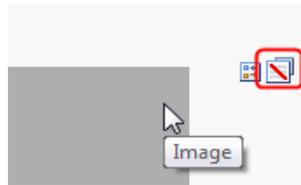
- Automatic Call Distributor
- Automatic Call Distribution
- Automated Call Distribution
- Automatic Call Director

## Design

This type of application is a test or quiz. In this example the score is determined and shown to the user immediately after taking the quiz. I used a few different techniques in this example, let's break it down.

This quiz is made up of some simple questions using radio lists. There is nothing special about the questions or the logic of them. The real magic comes in to play once the user submits their results. The quiz has three stages: start, score and end. The form is setup to “Display in the next stage” upon submission:

Each of the questions has an image associated to it that has information about the question and it will be shown to the user after they submit the form. To accomplish this, click the **Stages** tab and then the **Start** stage then click the button to hide the image in the stage:



The label that shows the user score is also hidden in the start stage using the same mechanism.

Now let's look at how the quiz auto-marks itself. In the **beforeSave** event there is some code that executes to mark all the questions:

```
app.getSharedData().getItem(form, app.getSharedData().processItem);
BO.F_TestScore.setValue(app.getSharedData().totalCorrect);
```

This example leverages the [group of functions used to process all the items in a page](#), which can be found on the DevWorks Wiki. Since these functions are described in that article I will not go over them here. The one that I will describe is the definition of processItem that I used:

```
app.getSharedData().processItem = function(item) {

    //only do this if you found a radiogroup (select one)...
    if(item.getType() === "radioGroup") {
        var curVal = item.getValue();
        var curID = item.getId();
        var keyVal =
app.getSharedData().getListObjectValueById(app.getSharedData().key, curID);
        //if the user answer is incorrect set the field to invalid and show the
correct answer
        if( curVal !== keyVal) {
            item.getBOAttr().setValid(false, "The correct answer for this question
is " + keyVal);
        } else {
            //add to the total score
            app.getSharedData().totalCorrect++;
        }
    }
}
```

```
}
```

When walking over all the items of the form, if we find a radiogroup (select one) then we get the current value and check it against the “key” that was setup in the Settings...Events...Custom Actions of the application. There are several different ways that you could implement something like this, but for simplicity sake I chose to create a simple object array where each object has an ID (the ID of the question) and value (the correct answer).

```
app.getSharedData().key = new Array({id:"F_SelectOne", value:"Both 1 and 2"},
{id:"F_SelectOne0", value:"Vector Directory Number"}, {id:"F_SelectOne1",
value:"Automatic Call Distributor"}, {id:"F_SelectOne2", value:"Service Level
Agreement"}, {id:"F_SelectOne3", value:"True"}, {id:"F_SelectOne4",
value:"Full-Time Equivalent"}, {id:"F_SelectOne5", value:"Both 1 and 2"},
{id:"F_SelectOne11", value:"Interactive Voice Response"}, {id:"F_SelectOne6",
value:"Average Speed of Answer"}, {id:"F_SelectOne7", value:"Local Area
Network"}, {id:"F_SelectOne8", value:"No"}, {id:"F_SelectOne9", value:"Real
Time Adherence"}, {id:"F_SelectOne10", value:"First Call Resolution"},
{id:"F_SelectOne12", value:"Key Performance Indicator"}, {id:"F_SelectOne13",
value:"Management Information System"});
```

I needed to create a helper function that when given an item's ID it would return the value from the “key” array. Here is the function that was created:

```
app.getSharedData().getListObjectValueById = function(theList, compareId) {
    var r = "";

    //loop all the items and return if you find the compareId
    for(var i=0;i<=theList.length;i++) {
        var theObj = get(theList, i);
        var theID = get(theObj, 'id');
        if( theID === compareId) {
            r = get(theObj, 'value');
            break;
        }
    }
    return r;
}
```

I chose to show the user the correct answer if they selected the wrong answer. This was a design decision and you could modify this if it does not fit your use case:

## Test Score: 4

Question 1

\* Schedule Adherence means

- Metric used in the call center to determine whether or not call center agents are working the amount of time they are scheduled to work.
- The degree to which agents stick to their schedulesSpecify Value 2
- None of the above
- Both 1 and 2

The correct answer for this question is Both 1 and 2



**What is schedule adherence and why is it **important** in the call center?**

Schedule adherence is the degree to which agents stick to their schedules. Considering that staffing is the single biggest cost facing any call center – and that every minute counts when it comes to meeting customer service levels – it’s easy to see why call center managers are putting an increasing emphasis on improving it and are looking to their workforce management software for solutions.

If you run a small call center with 5 to 20 agents, you might think schedule adherence isn’t all that important. But if you take a closer look at all the different junctures throughout the day where agent time is “lost,” you’ll realize that it can add up to a serious loss of revenue over the course of a year.

When the code is done walking all the items in the form is stores the total score in a global variable (`app.getSharedData().totalCorrect()`). So the second line of code in the **beforeSave** event copies that **totalScore** into a hidden field. Once that value is in the hidden field it will be displayed by the label that is setup to mirror the content of the field:

Test Score: **{Test Score}**

The hidden field will also insure that the total score appears in the database.

Once the form has been submitted, I use the following code in the **onShowActionButtons** event of the Form to hide the submit button so that it cannot be submitted any further:

```
if(BO.getCurrentStage() === "ST_Score") {
  var actionButtons = form.getStageActions();
  for(var i=0; i<actionButtons.length; i++){
    if(get(actionButtons, i).getId() === 'S_Submit1')
      get(actionButtons, i).setVisible(false);
  }
}
```

## Adapting to your Use Case

1. If your test contained different question types other than **Select One** then you would need to modify the

**processItem** function so that only the question types get processed.

2. If you need to process different types of questions differently from others then you could create different implementations of the **processItem** function and then call each accordingly.

3. If you need to add questions then you will need to add the UI for the questions, but also add the ID and correct value to the “key” array. If you don't like having to have the “key” as a JavaScript array, you could create another form in the FEB application that contains the question ID and correct value. Then you could use a search service to return the answer key and process them that way. The reason I chose not to do that is because then I would have to have a hidden table in the form that received the service return value, which would add more complexity to my form.

## Need Help?

If you have any questions regarding this sample please post to the [DevWorks Community Forum](#) and we would be happy to assist you!