

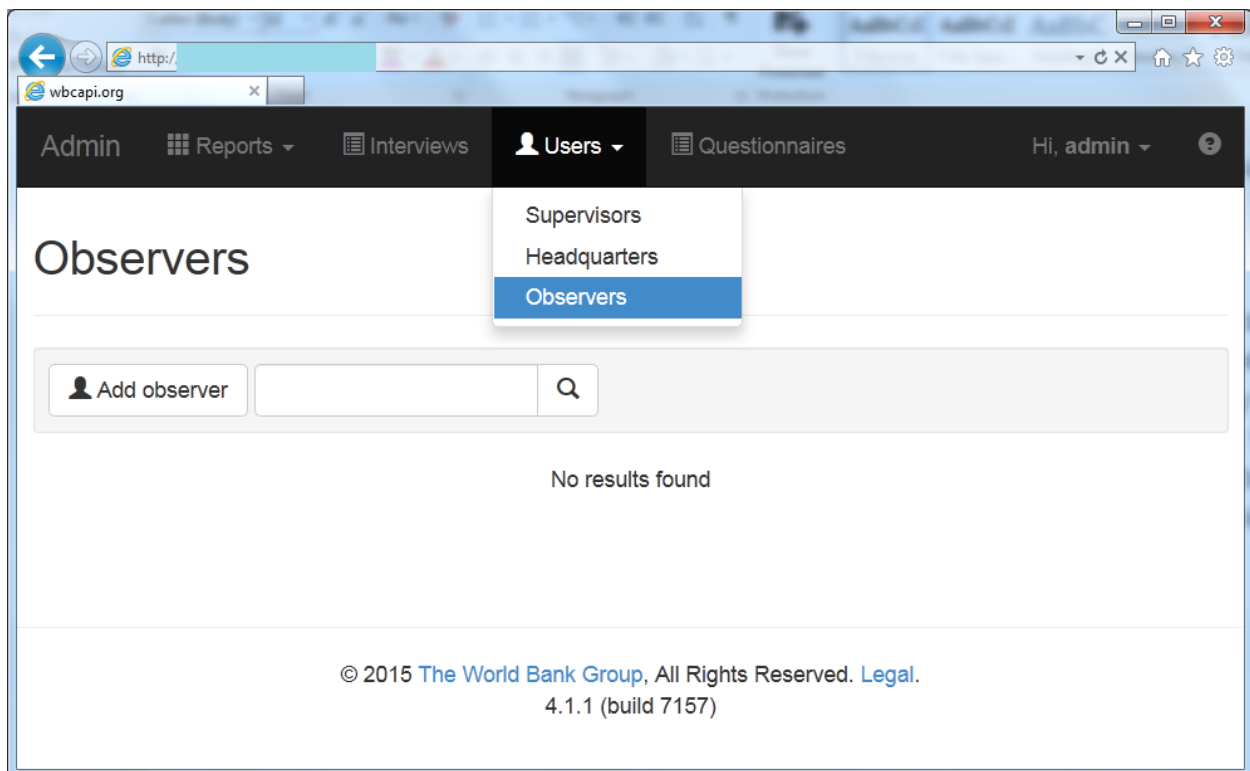


Dear friends of Survey Solutions,

Below is an overview of the new features added in version 4.2.0.

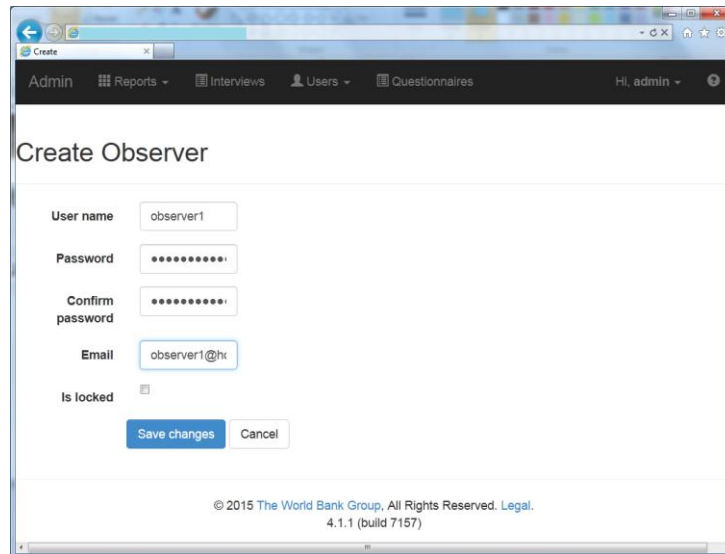
## Observer accounts

The new version adds a great feature for collaboration. An observer account can be added by the administrator user to let a certain person monitor the activity (shadow) another user in the system without being able to damage the system by unintentionally deleting or modifying an assignment. This feature improves the reliability of the system as it tightens the security: only those users that need to modify the data can be given the permission to do so, while other users, such as managerial personnel can be given a read only access for monitoring the progress.




To add a new administrative account the administrator should follow this procedure:

- 1) point to the **Users** item in the menu, select **Observers**;
- 2) a list of current observer accounts will be displayed (can be empty);
- 3) click **Add observer** button.
- 4) fill out the standard user account form specifying the login, password and contact for the user:



Click **Save changes** and the new account will be added to the list of the observer accounts.

When the observer logs into the system he will be able to select another user's account from the list and by clicking the  button view exactly what that user can view. Some operations are disallowed for the observer user, such as adding or modifying assignments and other operations that affect the data.

## Language extension: functions

Survey Solutions 4.2.0 adds a number of functions that help users write proper enablement and validation conditions. For example instead of

```
(education==4) || (education==5) || (education==9)
```

it is now possible to write

```
education.InList(4,5,9)
```

Here **InList()** is a function, which provides an alternative (shorter and more readable) notation for the above condition. Other functions provide functionality, which was not available or difficult

to write as a single expression. For example, the **GpsDistanceKm()** computes an approximate distance between the two locations on the Earth.

```
home.GpsDistanceKm(work)
```

The functions have a name and arguments. In the above example, **GpsDistanceKm** is the name of the function, and **home** and **work** are two of its arguments: the two locations the distance between which should be computed. This is more obvious in the alternative notation:

```
GpsDistance(home, work)
```

Note that some functions are written with a mandatory dot inside a function name, for example: **ZScore.Bmifa()**, which uses WHO child growth reference tables to compute the BMI-for-age z-score.

The users should pay attention to the types. Survey Solutions is using C# language for writing conditions, and in C# each function and each argument has a certain type, which indicates the kind of values the variable or function may take. All functions and their arguments are described in the function reference guide "**Survey Solutions Functions**". More functions to address common situations will be added in the future versions.

A common mistake that beginner users make is simply to call the function in say a validation condition. The condition requires a Boolean expression, kind of the  $x > 2$  that you wrote in the previous versions. Some of the functions are already providing the result as Boolean, for example `name.IsLike("Ch?n")` will be true for values "Chen" and "Chan", but some functions delegate the decision to the designer, who should decide how the result of the function affects the decision to switch a question on and off or consider a value valid or invalid. For example, a commute question may be enabled for respondents reporting their work is more than 10 km away from their home with a condition as: `home.GpsDistanceKm(work) >= 10.0`

## Language extension: addressing roster items

Another new feature added in Survey Solutions 4.2.0 is addressing the items in rosters by codes. From version 4.2.0 every roster item must have a code attached to it. Older questionnaires will continue to work and item codes will be generated in them by the system.

The situation when examining an item code comes frequently in the consumption and price surveys. Imagine for example the following case:

GROCERIES			<i>price</i>	<i>volume</i>
[0]	101	Milk	35.00	12
[1]	105	Toast	29.00	X
[2]	211	Honey	83.00	2

Here the groceries fixed roster inquires about the price and the volume of purchased items: milk, toast, and honey. However, the designer of the questionnaire wants to disable the volume question for the toast. This can be done by examining the item code in the enablement condition for the volume question:

```
@rowcode!=105
```

The condition refers to the system value **@rowcode**, which contains the code of the item in the row for which the volume is being evaluated. The @ character is intended. The effect of this expression is that it will evaluate to *true* for all items with codes different from 105 (and hence the *volume* question will be enabled) and it will evaluate to *false* for the item with code 105 (and hence the *volume* question will be disabled).

Naturally, the condition examining the item codes may be a part of a larger expression:

```
@rowcode.IsNoneOf(105,131,159,167) && (hhsz>3) && (price>2.2)
```

In this expression the question will be enabled when the item code is not one of the listed codes (105, 131, 159, 167) and the price of the item is above 2.2 (unspecified currency units) and the household size (hhsz) is more than 3.

Adding the item codes in the Designer is easy (item codes shown in green in the table below):

Roster Type: Fixed Titles

Variable name (?): groceries

Title: Groceries

Fixed roster titles: (?)

101	Milk	×
105	Toast	×
211	Honey	×

ADD TITLE