TD Mobile™

Quick Start Guide

Product Version 2.0

GUPTA

The software described in this book is furnished under a license agreement and may be used only in accordance with the terms of the agreement.

Last updated: September 2015

Legal Notice

Copyright © 2014-2015 Gupta Technologies, Inc. All rights reserved.

Gupta, Gupta Technologies, the Gupta logo, Gupta Powered, the Gupta Powered logo, ACCELL, Centura, Centura Ranger, the Centura logo, Centura Web Developer, Component Development Kit, Connectivity Administrator, DataServer, DBIntegrator, Development Kit, eWave, Fast Facts, NXJ, Object Nationalizer, Quest, Quest/Web, QuickObjects, RDM, Report Builder, RPT Report Writer, RPT/Web, SQL/API, SQLBase, SQLBase Exchange, SQLBase Resource Manager, SQLConsole, SQLGateway, SQLHost, SQLNetwork, SQLRouter, SQLTalk, Team Developer, Team Object Manager, TD Mobile, Velocis, VISION, Web Developer and WebNow! are trademarks of Gupta Technologies and may be registered in the United States of America and/or other countries.

SQLWindows is a registered trademark, and TeamWindows, ReportWindows and EditWindows are trademarks exclusively used and licensed by Gupta Technologies.

The product described in this document is distributed under licenses restricting its use, copying, distribution, and decompilation/reverse engineering. No part of this document may be reproduced in any form by any means without prior written authorization of Gupta Technologies Corporation and its licensors, if any.

THE DOCUMENTATION IS PROVIDED “AS IS” AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID. GUPTA TECHNOLOGIES, INC. SHALL NOT BE LIABLE FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES IN CONNECTION WITH THE FURNISHING, PERFORMANCE, OR USE OF THIS DOCUMENTATION. THE INFORMATION CONTAINED IN THIS DOCUMENTATION IS SUBJECT TO CHANGE WITHOUT NOTICE.

This document may describe features and/or functionality not present in your software or your service agreement. Contact your account representative to learn more about what is available with this Gupta Technologies® product.

Gupta Technologies, Inc.
1420 Rocky Ridge Drive, Suite 380
Roseville, CA 95661

www.guptatechnologies.com

Table of Contents

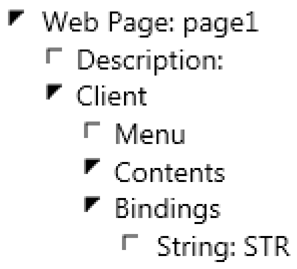
Introduction

This quick start guide is meant to help you become familiar with some concepts of TD Mobile™ to get you started quickly. To get the most from this document, read it in conjunction with using the samples shown below the explanatory text, which are indicated by [sample: xxxxx.apx].

Bindings

Bindings are the key concept in creating a TD Mobile app. They are the means by which you transfer data between the client and the server.

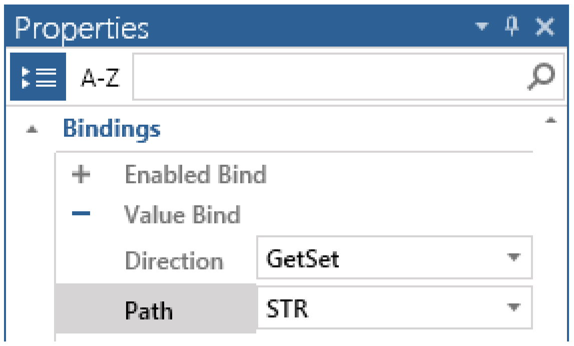
Defining a Binding is a lot like declaring a variable. You can find the Bindings section in the Client half of a page's Outline window:



```
◤ Web Page: page1
    ◸ Description:
    ◤ Client
        ◸ Menu
        ◤ Contents
        ◤ Bindings
            ◸ String: STR
```

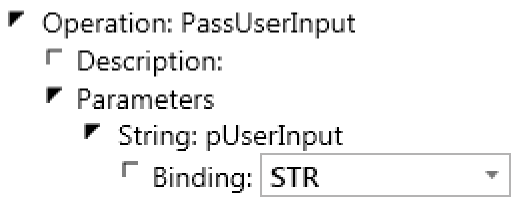
Using a Binding with a Data Field

Drop a Data Field (from the Layout tab of the ribbon) onto the Phone Layout. Attach a Binding for it in the Properties window for that Data Field:



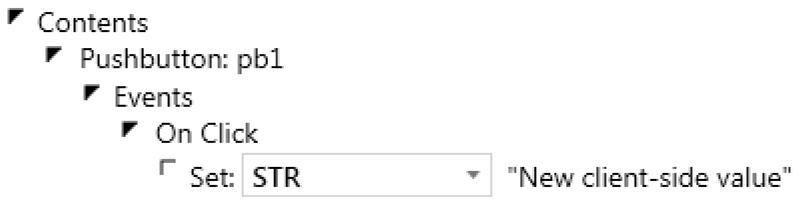
Direction: GetSet
Path: STR

Thereafter, any user text typed into that data field will update the Binding with the same text. To pass that user input to server-side code, you would specify that Binding as a parameter in an Operation:



```
Operation: PassUserInput
  Description:
  Parameters
    String: pUserInput
      Binding: STR
```

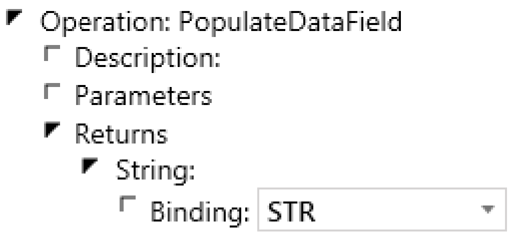
To set the value of the Binding in Client code, say as the result of a button click, use the Set statement:



```
Contents
  Pushbutton: pb1
    Events
      On Click
        Set: STR   "New client-side value"
```

The specified text would replace whatever text was in the data field.

To set the value of the data field with server-side code, specify its Binding as the Return value of an Operation:

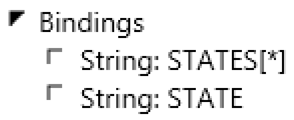


```
Operation: PopulateDataField
  Description:
  Parameters
  Returns
    String:
      Binding: STR
```

[Sample: HelloWorld.apx]

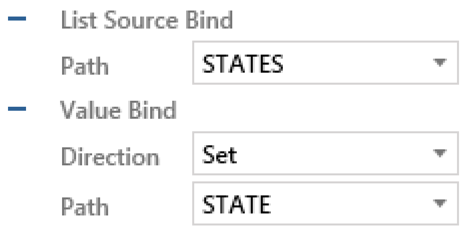
Using a Binding with a Combo Box

The Combo Box control is used to display a dropdown list of strings, from which the user can select one. One way to populate it is with a Binding that is an array of type String. To set the selected item or to capture the user's selection, you will also need a Binding that is a single instance of type String:

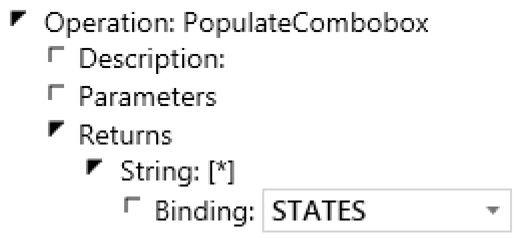


```
Bindings
  String: STATES[*]
  String: STATE
```

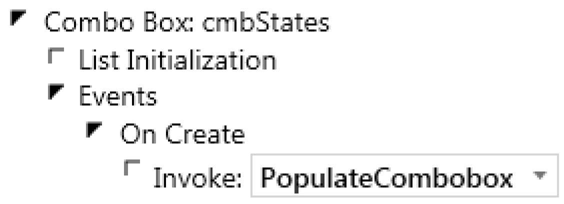
The array binding will be the List Source Bind for the combo box, and the single binding will be the Value Bind for it (as seen in the Properties window for the combo box):


List Source Bind
Path: STATES
Value Bind
Direction: Set
Path: STATE

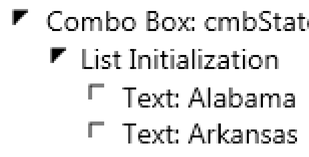
You can populate the List Source Bind of a combo box by returning it from a server-side Operation:


Operation: PopulateCombobox
Description:
Parameters
Returns
String: [*]
Binding: STATES

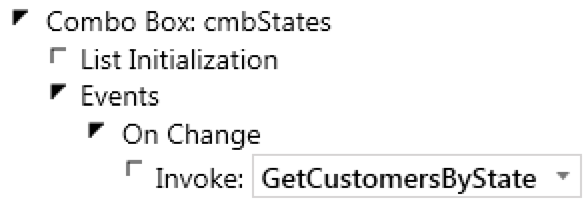
You can Invoke that Operation from any client-side Event, for example, the combo box's Create event:


Combo Box: cmbStates
List Initialization
Events
On Create
Invoke: PopulateCombobox

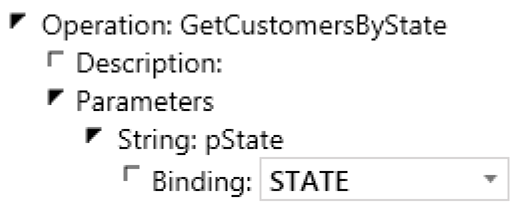
Alternatively, you can populate a combo box without a List Source Bind, by manually typing the strings in the combo box's List Initialization section:


Combo Box: cmbStates
List Initialization
Text: Alabama
Text: Arkansas

In either case, when the user selects an item in the combo box, the Value Bind will be populated with the selected string. The selection itself will trigger the combo box's Change Event, where you might call some server-side code:


Combo Box: cmbStates
List Initialization
Events
On Change
Invoke: GetCustomersByState

And pass that selected Bind to the Operation as a Parameter:

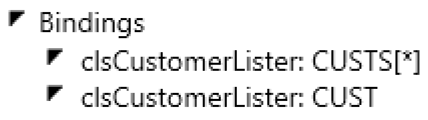


Operation: GetCustomersByState
Description:
Parameters
String: pState
Binding: STATE

[Sample: ComboBoxBinding.apx]

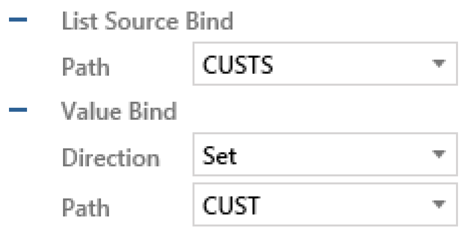
Using Bindings with a List View Control

The List View control is used to display a list of information, or rows of data. You populate it with a Binding that is an array of a class or of a UDV. If you want the user to be able to select a row or an item in the list, you will need a Binding that is a single instance of that same class:



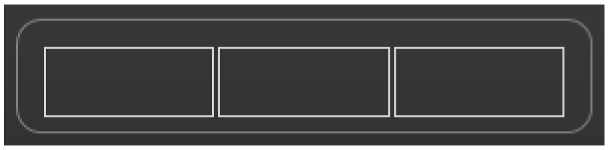
Bindings
clsCustomerLister: CUSTS[*]
clsCustomerLister: CUST

The array binding will be the List Source Bind for the list view control, and the single binding will be the Value Bind for it (as seen in the Properties window for the list view control):

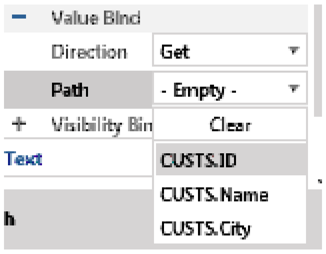


List Source Bind
Path: CUSTS
Value Bind
Direction: Set
Path: CUST

Put a Layout Grid inside the List View control. Configure the columns and rows of the Layout Grid (in the Properties window) depending on what data you are going to display for each item in your list:

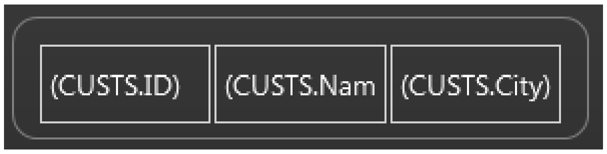


Drop Text controls (from the Layout tab of the ribbon) into the cells of the Layout Grid. Set the Value Bind for each. The options available will be the fields in the List Source Bind you specified for the List View control:

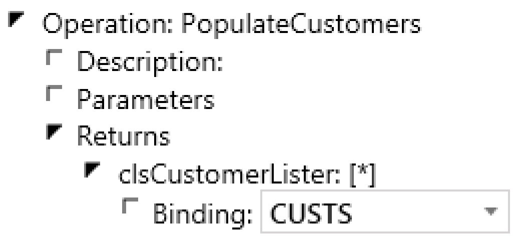


Value Bind
Direction: Get
Path: - Empty -

You design the List View as if for the first item in your list. All items will share that layout:

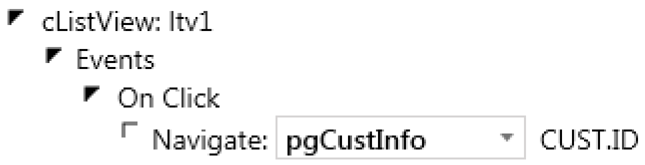


Populate your List Source Bind from server-side code by making it the Return value of an Operation:



```
Operation: PopulateCustomers
    Description:
    Parameters
    Returns
        clsCustomerLister: [*]
            Binding: CUSTS
```

When the user selects an item in the list, that item will become the Value Bind (in our example, CUST). You can then, on that selection Click, use that Value Bind, or one of its fields—for example, CUST.ID—as the parameter of an Operation you invoke, or as the parameter of another Page you might navigate to:

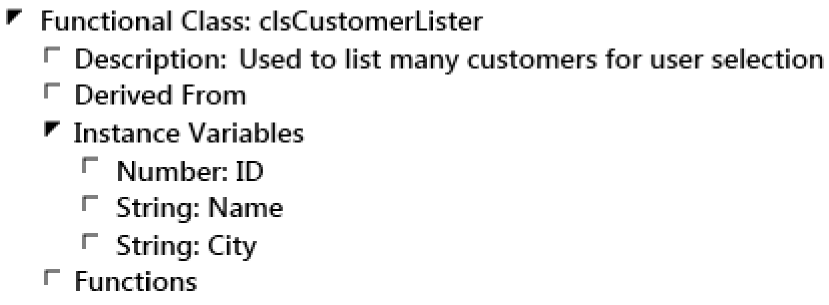


```
cListView: ltv1
    Events
        On Click
            Navigate: pgCustInfo    CUST.ID
```

[Sample: ListViewBinding.apx]

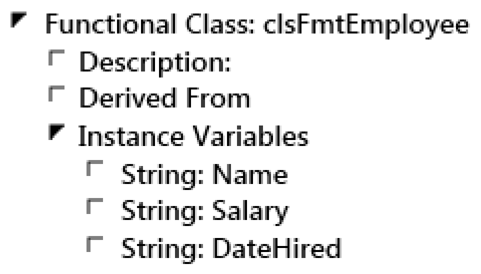
Defining Classes in TD Mobile (with Bindings in Mind)

You could have a Customer class with the 20 or 30 fields you need to fully define that entity. But say you want to show a list of customers in a List View, and only plan on populating the list with three fields: CustomerID, Name, and City. It would be inefficient to use an array of type Customer to send an overly large, mostly null-filled array across the internet. It would make sense then to define a class specifically for the binding you will need to populate that List View; for example:



```
Functional Class: clsCustomerLister
    Description: Used to list many customers for user selection
    Derived From
    Instance Variables
        Number: ID
        String: Name
        String: City
    Functions
```

If you are planning to format numeric and date fields from your class, you might want to define them as strings, if the formatted string version of your number or date is what you want to display (and so is the field you would want to Bind to).


- Functional Class: clsFmtEmployee
  - Description:
  - Derived From
  - Instance Variables
    - String: Name
    - String: Salary
    - String: DateHired

Depending upon your needs, you might want to define both a Number: nSalary for mathematical needs and String: sSalary for your formatted display string.

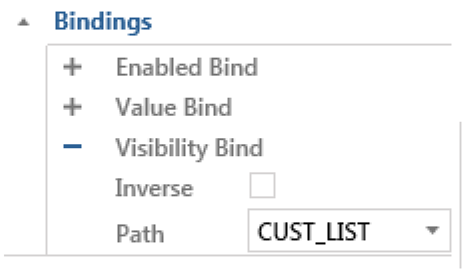
[Sample: FormattingAndClasses.apx]

Bindings as Booleans

All types of Bindings can be used and evaluated as Booleans. There are default evaluations for all types; for example a String Binding is True if it is not empty and False if it is; a custom type (class or UDV) will be True if not null and False otherwise; an array is True if it has items and False if it does not. You can also define what makes your binding True or False (check out the Bindings tab on the IDE ribbonbar). Using Bindings as Booleans is useful in many ways.

Hiding/Showing Controls with Visibility Binds

Let's say you were populating a List View control with a Binding named CUST_LIST of type clsCustomerLister, and you wanted to have column headers above the lister. You could drop a Layout Grid right above the List View, set it to have three columns and one row, drag text fields into each cell, giving each an appropriate caption, and setting the Visibility Bind for each text field to CUST_LIST:


Bindings
- + Enabled Bind
- + Value Bind
- − Visibility Bind
  - Inverse ☐
  - Path: CUST_LIST

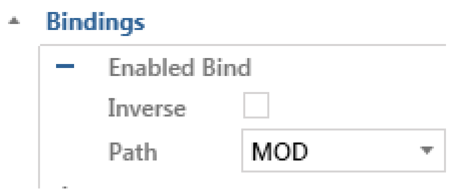
When the customer list is populated, it will show, and the headers above it will show. When the list is empty, neither the list nor the headers will show.

[Sample: VisibilityBinds.apx]

Enabling/Disabling Controls with Enabled Binds

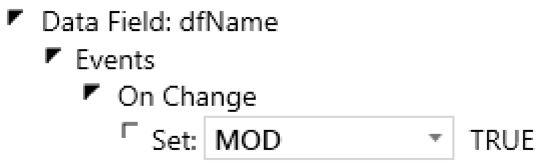
Let's say you had three Data Fields you populated with text from your database and presented to the user. You are using data fields instead of Text fields because you want the user to be able to edit the text. Below the data fields you have a button with the caption Save Changes. You uncheck the button's Enabled property in the Properties window so that it starts off disabled; you want to enable it only if the user changes some of the text.

Define a Binding of type Boolean, call it MOD. Set it as the Enabled Bind for the button:



```
Bindings
  Enabled Bind
    Inverse  [ ]
    Path     MOD
```

In each of the three data fields' On Change events, add this code:



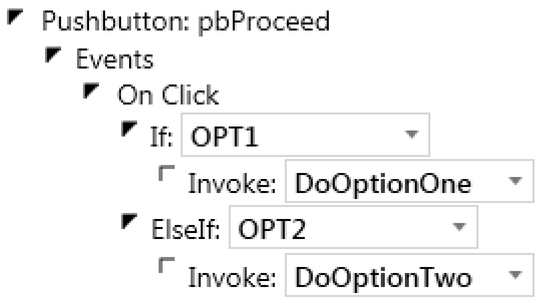
```
Data Field: dfName
  Events
    On Change
      Set: MOD  TRUE
```

Any change to any of the data fields will enable the button.

[Sample: EnableBinds.apx]

Controlling Code Flow with Boolean Binds

You can use Bindings as the Boolean expression of If statements in client-side code. In this example, OPT1 and OPT2 are bound to radio buttons, and the code flow on the button click will be dictated by the radio button selected by the user:

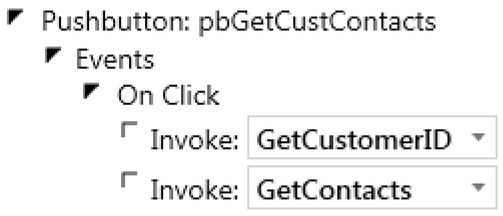


```
Pushbutton: pbProceed
  Events
    On Click
      If: OPT1
        Invoke: DoOptionOne
      ElseIf: OPT2
        Invoke: DoOptionTwo
```

[Sample: CodeFlowBinds.apx]

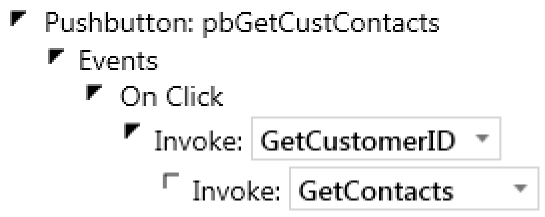
Timing in Client-Side Code

Say you were going to call an Operation named GetCustomerID that would set the value of a Binding named CUSTID. You want to use that CUSTID to look up the contacts for that customer with an Operation named GetContacts. You might code the Click Event Handler for a pushbutton like the following:



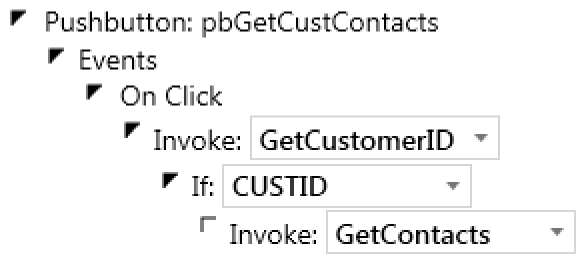
```
Pushbutton: pbGetCustContacts
  Events
    On Click
      Invoke: GetCustomerID
      Invoke: GetContacts
```

That would likely be a problem. The nature of JavaScript is such that those calls would run immediately, the second not waiting for the first to finish. As a result, you might not have the correct Customer ID when you call GetContacts. TD Mobile allows you to control this timing. If you indent the second call, it will not run until the call above finishes and returns:



```
Pushbutton: pbGetCustContacts
  Events
    On Click
      Invoke: GetCustomerID
        Invoke: GetContacts
```

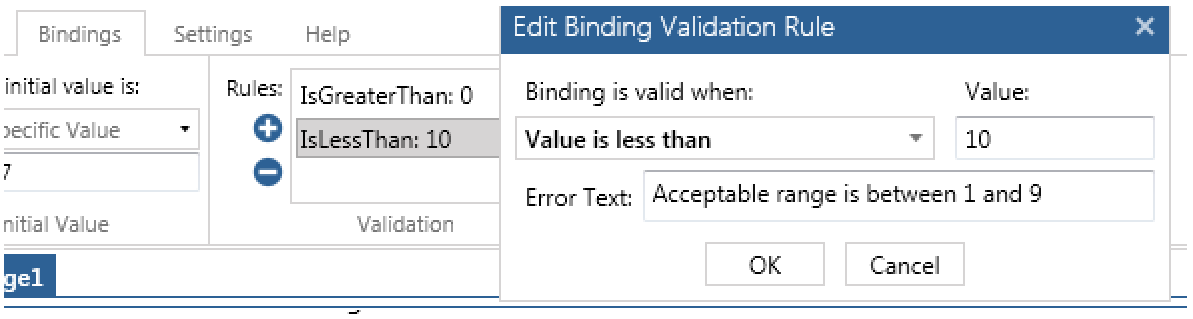
That's better. But, if you made sure GetCustomerID returned a CUSTID that would evaluate to True only if it succeeded in getting the ID, you could wait for it to finish *and* ensure that it succeeded:



```
Pushbutton: pbGetCustContacts
  Events
    On Click
      Invoke: GetCustomerID
        If: CUSTID
          Invoke: GetContacts
```

Validating User Input with Bindings

You can validate fields by defining Rules to the Bindings that are attached to those fields. In the Bindings tab of the IDE ribbon, you can add, edit, or remove Rules:



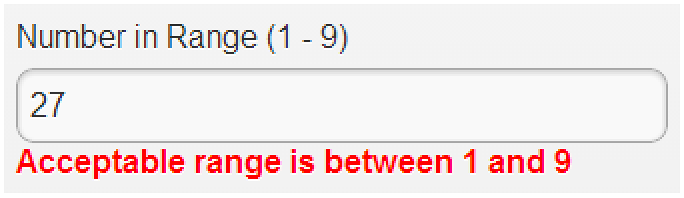
Edit Binding Validation Rule

Binding is valid when: Value is less than

Value: 10

Error Text: Acceptable range is between 1 and 9

The Rules include Required (cannot be blank), greater than, less than, equal to, and so on, and you can write a custom message for each rule that will display below the field when the rule is not met:

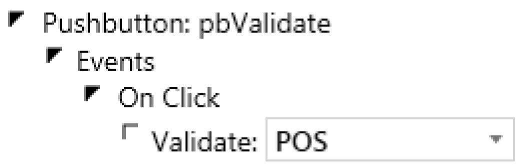


Number in Range (1 - 9)

27

Acceptable range is between 1 and 9

Rules are enforced when you try to use the value of a bind by passing it as a parameter to an Operation. You can also force it by using the Validate statement in client-code:


```
Pushbutton: pbValidate
  Events
    On Click
      Validate: POS
```

[Sample: Validation.apx]

Database Connectivity

It is strongly recommended that you connect to your database with the TD.NET function SqlConnectDotNet(), instead of the old SqlConnect(). SqlConnectDotNet uses a connection string instead of the System Variables SqlDatabase, SqlUser and SqlPassword. Read more about this recommendation in the TD Mobile Help section Database Connectivity.

Communicating with your database otherwise should work in TD Mobile just as it does in Team Developer. The same Sql* functions, like SqlPrepareAndExecute() and SqlFetchNext(), are available to you. Sql Handle is an available variable type.

The one warning here, though, is if your Sql Handles are stored in global variables and left connected. Those, like all global variables in TD Mobile, are stored by IIS as Session objects, and these will be lost when the Session times out. In that case, you will have to reconnect them, and the previous connections may be left open in your database. Therefore, you should connect and disconnect after each database interaction, using local variables for your Sql Handles.

If it is necessary to use Sql Handles as global variables, see the “Extending Idle Timeout” section, which follows.

Global Variables

When you are using global variables in your TD Mobile application, you need to be aware that the variables will be stored in an IIS Session object. When that Session times out (by default, after 20 minutes of idleness), the variable will be destroyed and its value lost.

Therefore, use them sparingly and be prepared to repopulate their values as necessary. The Session timeout can be extended if necessary.

Extending Idle Timeout

The default timeout, due to idleness, for Session State and for the Authentication cookie, is 20 minutes. This is so the server is not over-burdened with keeping in memory many instances of abandoned applications. In some circumstances—for example, when there is a very small pool of users—you might want to ask the server to keep this memory through longer spans of idleness. To do this, you must change the following configuration settings in the Internet Information Services (IIS) Manager:

- Set the Application's Session State timeout to the desired number of minutes.
- Set the Application's Authentication (Windows Forms) timeout to the desired number of minutes.

- Set the IIS Application Pooling timeout (this is not an application level setting) to zero; in effect, turned off.

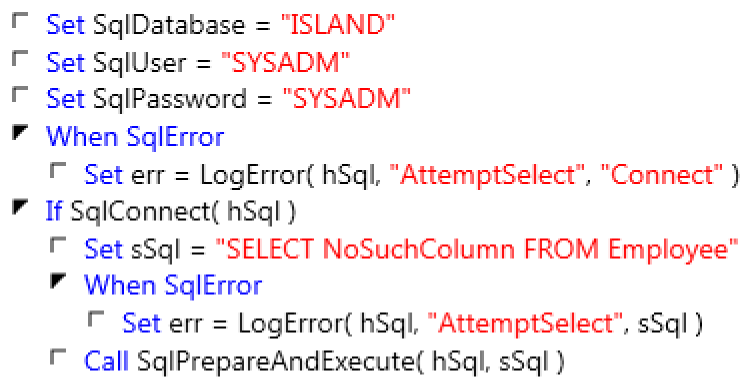
The first two, alternatively, could be set in the application's web.config file with the following manual entry:

```
<system.web>
…
<authentication mode="Forms">
<forms timeout="180" />
</authentication>
<sessionState timeout="180" />
</system.web>
```

SQL Error Handling

If you don't do anything to handle SQL errors, you will get a runtime message box that gives you information about the error, much like in Team Developer.

Be aware, however, that there is no Application Action where you can receive SAM_SqlError messages to capture all errors. To trap SQL errors you must use the 'When SqlError' statement local to any SQL for which you want to provide error handling. Within the When SqlError block you can call a central function to handle error processing, or write the error information to a log, or whatever you would normally do to handle a SQL error:



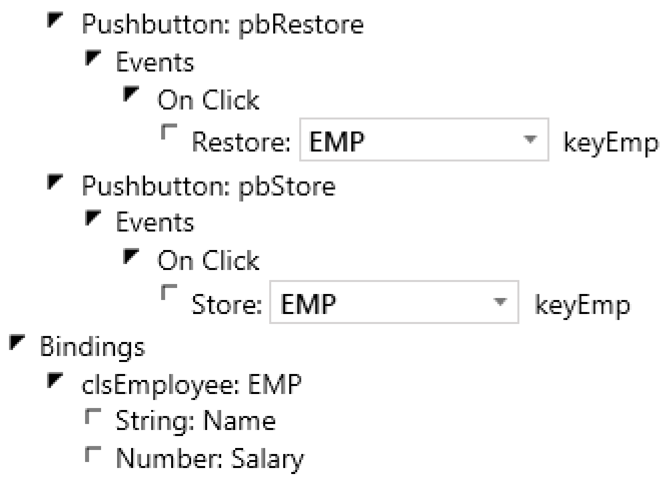
```
Set SqlDatabase = "ISLAND"
Set SqlUser = "SYSADM"
Set SqlPassword = "SYSADM"
When SqlError
    Set err = LogError( hSql, "AttemptSelect", "Connect" )
If SqlConnect( hSql )
    Set sSql = "SELECT NoSuchColumn FROM Employee"
    When SqlError
        Set err = LogError( hSql, "AttemptSelect", sSql )
    Call SqlPrepareAndExecute( hSql, sSql )
```

[Sample: SqlErrorHandling.apx]

Store and Restore

The Store and Restore actions can be used in client-side code to store data in the browser's database. Anything you can store in a Binding, you can store this way, even if it is a custom class in your app. You can specify how long the storage should last. Select the Binding and in the Local Storage section of the Bindings tab on the IDE ribbonbar.

The following examples shows a Binding named EMP which you Store it and Restore it with a key you named "keyEmp":



```
◤ Pushbutton: pbRestore
    ◤ Events
        ◤ On Click
            ◸ Restore: EMP  keyEmp
◤ Pushbutton: pbStore
    ◤ Events
        ◤ On Click
            ◸ Store: EMP  keyEmp
◤ Bindings
    ◤ clsEmployee: EMP
        ◸ String: Name
        ◸ Number: Salary
```

Imagine that you have Data Fields on your web page that are each bound to various fields from the clsEmployee class. As you type into the fields, the Binding is updated with the information. You click the Store button (pbStore) and the current state of the binding is stored in the browser database. You navigate away from the page and come back to it later. Because your web page is innately stateless, the fields are blank. But you click the Restore button and the Binding is restored and your fields display the data you had entered.

[Sample: StoreAndRestore.apx]

Using Link Controls

Link controls, when clicked, will link to a separate application on your device and use the value you give it to act on that application. You specify the Link Type in the Properties window. You can specify the value for the link either in the Link URL field in the Properties window, or in its Value Bind. The value will show in the display unless you provide a Caption that shows instead. For example, if you clicked an Email Link with a value of an email address, your email program would start up a new message ready to send to that address. Other Link Types are Web, Phone, Map, SMS, and JavaScript.

[Sample: Links.apx]

Importing an External Library from Team Developer

You can include any *.apl file from Team Developer by adding it to the Files section in the Application window of the IDE. There are several things to be aware of:

- The .apl file must be first saved in Text format.
- Unsupported SAL code must be removed or replaced. These will be revealed as compile errors in your application. These include:
    - Classes, variables, or parameters that are Window Handle types
    - Functions that operate on or require Window Handle types, including the following function groups: SalGetWindow*, SalGrid*, SalList*, SalPic*, SalReport*, SalMeter*, SalBtn*, SalDlg*, SalDragDrop*, SalEdit*

Be aware that when you make changes to that library from your TD Mobile app, and then save your app, those changes will be saved in the .apl file as well. It is not necessary to open the .apl file in Team Developer to change the library.

FAQs

Q: Can I use a List View for data input, like a table window or grid?

No, the List View is meant to display data and allow item selection.

Q: What kind of controls can I put into a List View Layout Grid?

Right now, only a Background Text, an Image or a Link. This list could grow in the future.

Q: Can I arrange the display in a List View to accommodate different size fields?

Yes, you can use the Column Widths property of the layout grid to have different size columns. The property takes a comma-delimited list of percentages (without the percent sign); for example, "40,20,25,15". The list count should match the number of columns. The values should add up to 100.

Q: Can I use a UDV (a custom class) as a Page Parameter?

Yes, that is supported. Be aware, though, that some browsers impose a size limit on query strings. Depending on the size of the UDV, you might want instead to use a global variable that would also be available to the new page.

Q: Can I call SalQuit()?

No, that would cause a compile error. A TD Mobile app is "quit" when you close your browser or navigate away from your app's pages. If you authenticate users by calling SalWebLogin() , you can log them out with a call to SalWebLogout().

Q: Can I write out to a file or log Trace information on the server?

Yes. SalFile* and SalTrace* functions will work in server-code, but the IIS Application Pool account must have I/O permissions for the folder you are writing to.