TD MOBILE™

Primer

Product Version 2.0

GUPTA

TD Mobile™: Primer, Product Version 2.0

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 10, 2015.

Legal Notice

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

Gupta Technologies.com

Table of Contents

What is TD Mobile?

TD Mobile is a development system for building mobile cross platform enterprise apps that you can access from a phone, a tablet, or really from any internet-connected device that has a browser. TD Mobile apps are 100% cross platform, let you integrate device features such as a GPS sensor, the camera and many other device apps. For mobile enterprise apps data access to enterprise systems is a key feature. TD Mobile offers no-coding enterprise database access to SQL and NoSQL databases. If you need to go beyond database access TD Mobile includes a .NET programming language that allows you to implement virtually any backend logic you might need.

Easy design and development of the frontend app

Designing the pages of your TD Mobile app is a snap. Just drag the objects you want to have on your page from the ribbon bar onto the page. Choose from a variety of predefined themes for your look, or design your own themes altogether. Define the details of your screen objects using the TD Mobile property pane.

In mobile applications, developers program the behavior of the web pages and their controls using JavaScript. TD Mobile uses Event Actions, which are specific commands (Based in JavaScript) that do something commonly required on the client side so the developer does not have to write the JavaScript code.

Leverage device features like the current GPS position, taking a photo or video, accelerometer, phone links, sms links, email links and maps links. Just a tap will start the native device app to, for example, show the current GPS position on a map or send a text message. Read barcodes and take signatures for further processing.

TD Mobile includes many powerful controls, from datafields, comboboxes, grid controls to powerful charting and map controls. And all the controls can be easily bound to enterprise data sources.

Binding screen objects to data sources Is a snap. Bindings are used to tie data to the GUI components in a mobile enterprise application. Once the data models are defined, developers can easily tie fields on those models to fields in the UI without writing any code. TD Mobile automatically generates the service interfaces and client side logic to glue everything together.

Here is the amazing thing about TD Mobile: with it you can create state-of-the-art mobile web apps that use the latest technologies – HTML5, CSS3, JavaScript – *even if you have little or no knowledge of those technologies*.

If you need to dig deeper into JavaScript or HTML programming you can always choose to do so in writing your own JavaScript functions and your own HTML controls.

Easy enterprise backend integration

TD Mobile offers no-coding SQL and NoSQL database access. Data connections pull data from a backend data source such as a relational database or a NoSQL database like MongoDB. In TD Mobile, data connections are defined graphically so the developer does not have to write any code or learn the query syntax of the database they want to use. The data connection requires a few easy steps using TD Mobile.

Create, read, update and delete (crud) operations can be implemented without any programming using the new data classes and data operations. The new data classes and data operations work for SQL and NoSQL databases. XML data storage support and a programmable custom interface add to the power of this time saving new feature.

Integrate enterprise software solutions or your SOA architecture via Web Services. Integrate your mobile enterprise apps with existing software solutions like SAP or other solutions that provide a Web Services interface. TD Mobile gives you the power to integrate all your diverse systems into one easy to use mobile app.

And again if you need to go deeper than the standard crud database operations you can always leverage the power of the SAL .NET programming language to build any backend functionality you need including access to the .NET framework.

Deploying your apps is easy as well. Deploy your app to a Windows Server that runs Internet Information Server and your app is live.

If you are one of those developers who is not already a specialist in HTML5, CSS3, JavaScript, this document is for you. Its intent is to show you what you need to learn, and in a sense, what you don't need to learn. If you are an experienced software developer, you will find it very easy to use TD Mobile.

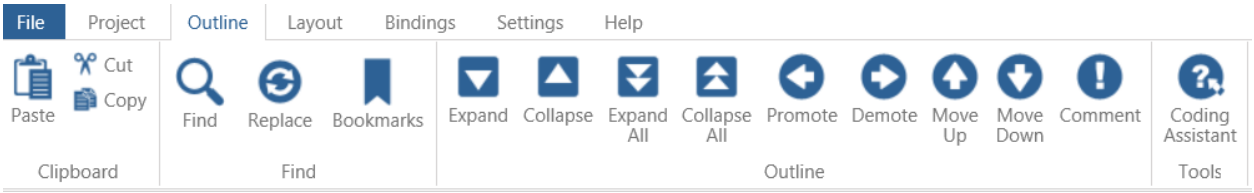
Let's get started.

Using the TD Mobile Outline

The first thing you notice about the TD Mobile outline is its structure. It is largely pre-built, and as you begin to add things to it, you will see that what you can add, and where, is pretty firmly enforced, and yet, in a helpful way. Here is the Outline window of a brand new, empty web page:

You've probably seen this tree-view type of structure before. The topmost node is the Web Page. Three nodes are indented one level under it: a Description section, a Client section and a Server section. You might say the web page contains these three sections. Notice how the icon for the Web Page is a filled-in triangle (meaning it has nodes indented below it, it contains something) and the icon for the Page Events node is an empty triangle (it does not contain anything yet).

The Server node is in a *collapsed* state, where we cannot see its contained nodes, but we can tell by the filled-in triangle that it does contain something. The Client section, in contrast, is in an *expanded* state. We can toggle between these states simply by double-clicking the nodes. Or if we have a complicated multi-nested group of nodes we can choose Expand All or Collapse All from the Outline tab of the Ribbon Bar:

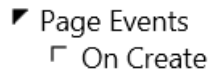


Some of the other options for changing the state of nodes in the outline, like Promote/Demote and Move Up/Move Down will only work if appropriate for the node. You will not be able to "promote" the Client node up to the same level as the Web Page, that wouldn't make sense.

What can be added into any specific section is presented to us in a helpful list. Here are a few different ways to see the list:

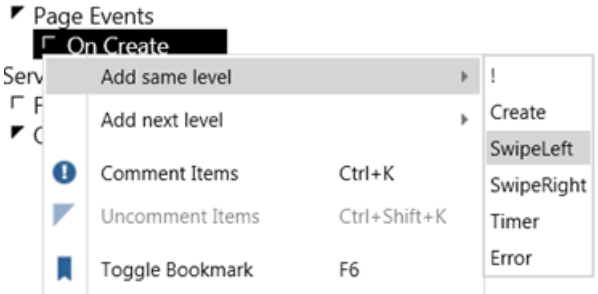
Simply click on an empty node to insert a node under it. Here we click on the Page Events section of our web page and we see a list of the events our code can respond to:

When you make your choice from this list, the event handler is inserted within the Page Events section:

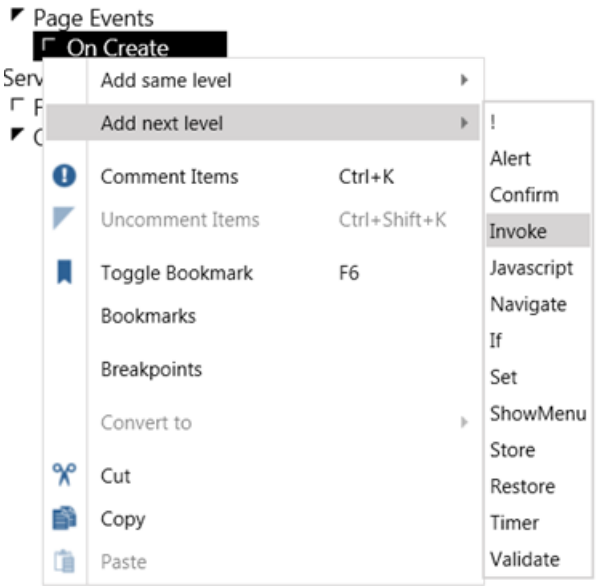


You could double-click that On Create node to see what can be inserted within it:

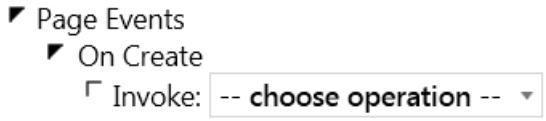
Or, instead, you could *right*-click the On Create node, and see that we can insert a node beneath it, as in the above screenshot, or we can insert a node with it, at the same level. If we select "Add same level", we see the options available for adding a *sibling* node, that is, another Page Event we could add:



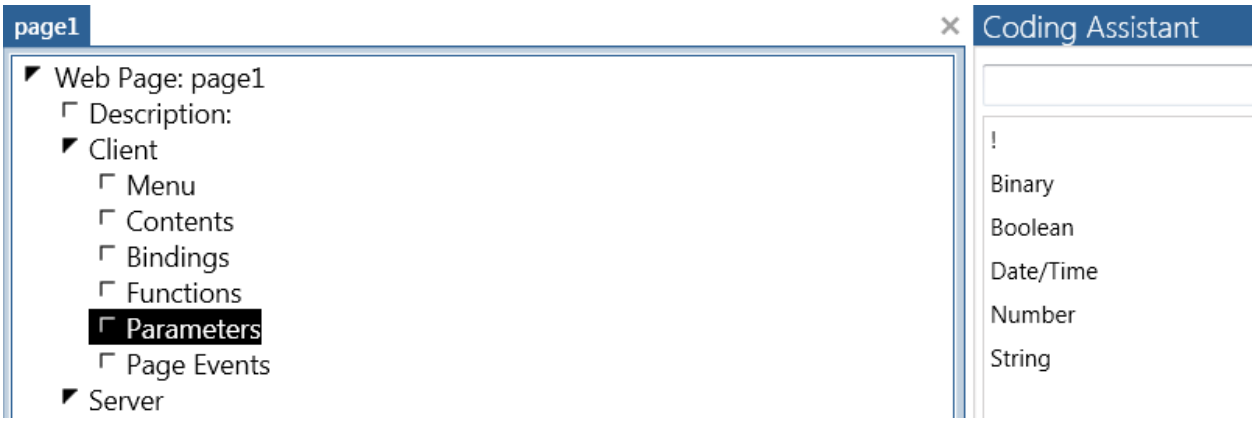
If we right-click that same node and select "Add Next level" we again see the options available for *child* nodes, that is, the commands available for responding to that event:



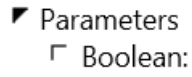
If we select the Invoke command, it would be inserted within the On Create event handler:



Another way to go is to use the wonderfully helpful Coding Assistant. (By default the Coding Assistant is shown in the rightmost pane of TD Mobile whenever you are in the Outline. If for some reason it is not showing, open the Outline tab of the Ribbon Bar and click the Coding Assistant icon in the Tools group). Below we have selected the page's Parameters section, and in the Coding Assistant we see the data types currently available for any parameters we want to declare:



If you double-click one of the data types there in the Coding Assistant, it will, in this case, declare a page parameter for you; all you need to do is give it a name:



The Coding Assistant will show both Same Level and Next Level options, if appropriate for the selection.

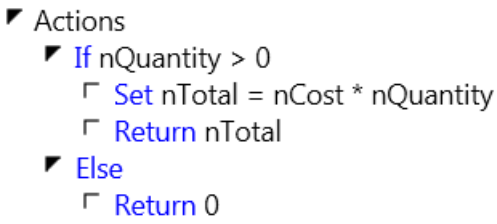
My main point here is that the TD Mobile Outline will always help with a list of available choices, and won't let you put

things in the wrong place.

Indentation

In these non-code sections of the outline, in this tree-view type of structure, indentation signifies a certain parent-child relationship. In code it means something else.

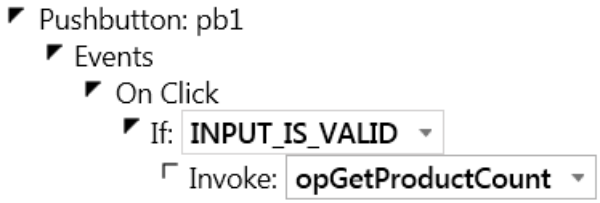
In server-side code, an indented line of code will only run if its parent line of code is a control flow statement that will evaluate an expression *and* if that expression evaluates to True. (By control flow statements I mean **If**, **While**, **Loop** statements and the like; more about those later in the Control Structures section of this document.) In the following example, if the parent line of code (If nQuantity is greater than zero) is true, then its two child lines will execute; if not the code indented under the Else would execute:



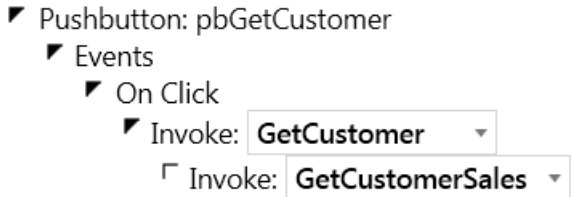
Notice there is no end word or symbol for the If; if the parent line evaluates as true, all the child lines indented under it will run.

Server-side code will always be in the Actions section of an Operation or a Function, while client-side code, or script, will always be in an event handler.

In client-side script, the indentation works the same way. This button click will only Invoke opGetProductCount if INPUT_IS_VALID is true:

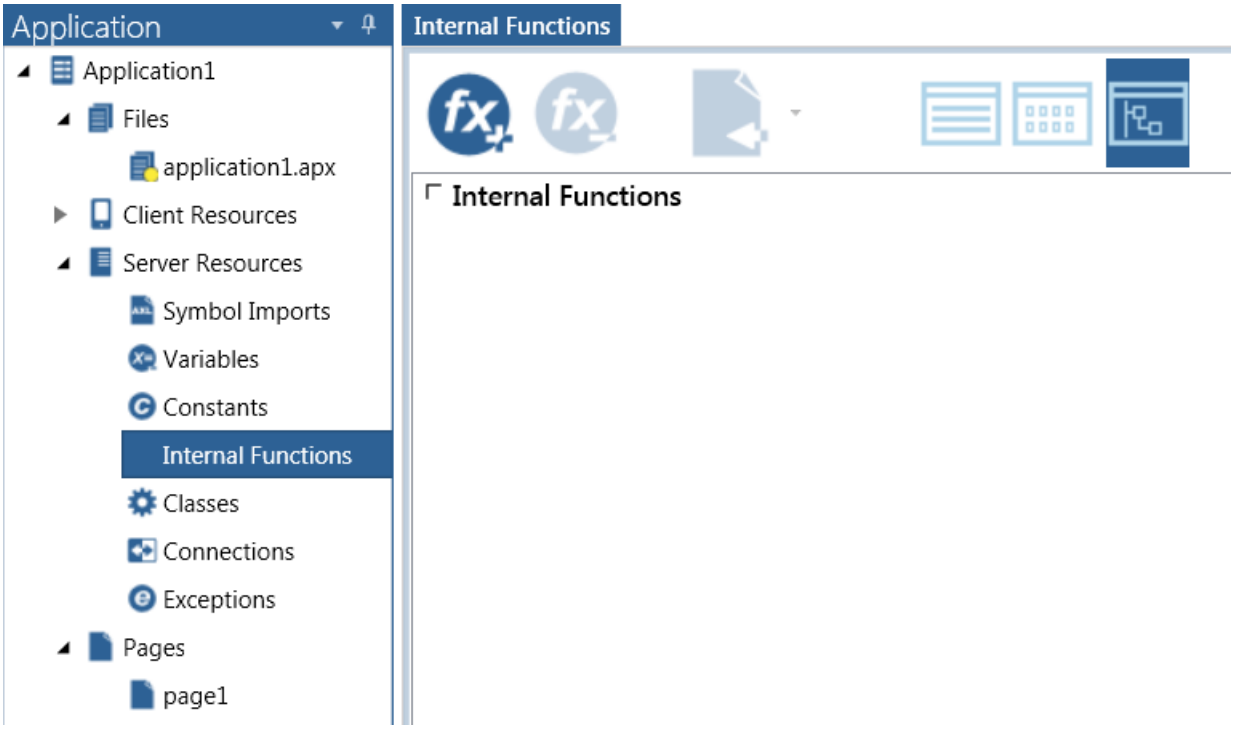


There's one other meaning that indentation can have in client-side script. When you invoke an Operation, it runs asynchronously; meaning that any same-level line after that Invoke will execute immediately, without waiting for the operation to finish. But sometimes, you *want* to wait till it finishes. You can make that happen with indentation. In the example below, GetCustomerSales will not be invoked until GetCustomer is completed.



The Application Window

So far we've been looking at the Outline window for a web page. Your app will likely contain many pages, and those pages will share resources. Shared resources, including the Pages collection itself, are accessible through the Application window in TD Mobile:



Above we are looking at the **Internal Functions** section of the **Server Resources** group; this is for global functions that can be called from anywhere in the application. Notice the icons at the top of the Internal Functions window. All of these Resources sections will have a similar set of icons. You can add items, like functions, by clicking the icon with the plus sign ("+"), and delete items by selecting the item and then clicking the icon with the minus sign ("-"). The three rightmost, squarish icons allow you to look at the contents in different views: List View, Tile View and Tree View, from left to right. I have Tree View selected in the above image, which is most like the web page outline we looked at earlier. Some sections, for example **Connections**, are specially constructed and do not have a tree view. But try all the views and see which you like.

Like in the web page outline, there is a key distinction between Client and Server. Within the Client Resources section, if you *do* know something about CSS and Javascript, here you can add files that your app can use. Within the Server Resources section you can create global variables, constants, functions, classes and more. You can read more about these things in the Help documentation within TD Mobile. (It really is worth your while to read the main documentation for TD Mobile: on the Help tab of the Ribbon Bar, click the Help button. Put it on your To-do list.)

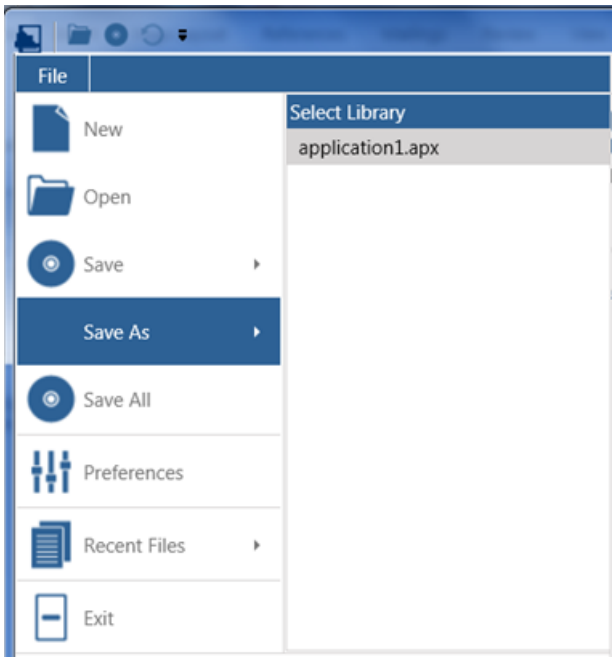
Your First TD Mobile App

It's a tradition that the first app you write in a new language should output the text 'Hello World!" So let's do it. Ours will not simply display output, but will show the basic wiring of client-server communication in TD Mobile. We'll take it step by step.

Open up TD Mobile. If you don't see a convenient TD Mobile icon to click, find the executable (Gupta.TD.IDE.exe) in the folder where you installed TD Mobile and run it.

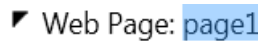
A new application is loaded into TD Mobile with a default name of application1.apx and with a single web page named page1. Let's change these.

Click on the File menu on the Ribbon Bar; select Save As; then click on application1.apx, as in the screenshot below:

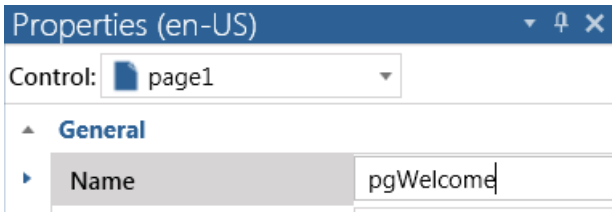


That will bring up the Save As window. Create a new folder wherever you like and name it 'HelloWeb'. Open that new folder then give your app a name 'HelloWeb.apx' and save it in there.

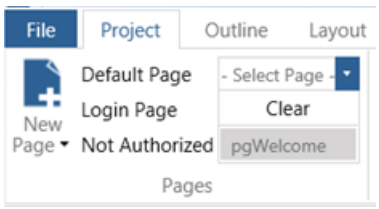
In the Application window of TD Mobile, make sure page1 is selected. In the page1 window, select the Outline tab on the bottom to bring up the code window. Select the text of the Web Page name:



Then type over page1 and rename it "pgWelcome". Alternatively, in the Phone Layout tab for page1, you could have changed the page's Name in the Properties window:

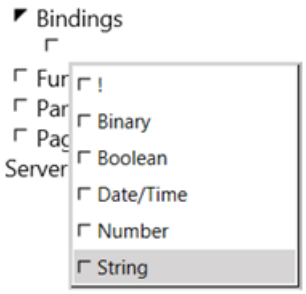


By default, page1 was the Default Page for your app, so make sure to specify this new page name as the Default Page: in the Project tab of the Ribbon Bar, click the down arrow for the Default Page and select 'pgWelcome':

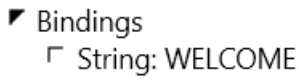


Now double-click on the Client section to expand it:

Within the Client section, double-click on the Bindings section and select the String data type:



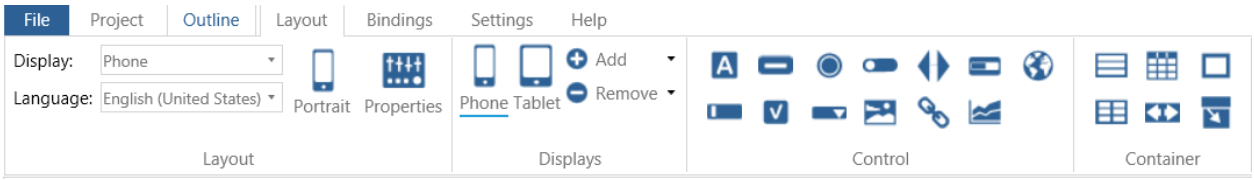
Give your new Binding a name, like you would name a variable. We have been using a convention of using uppercase for Bindings, but that is up to you. For now name it "WELCOME":



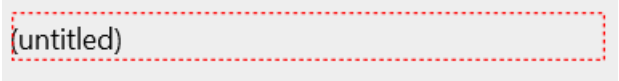
Bindings are very important in TD Mobile. They are the means through which data is passed between the client and the server. This will be a very simple example of how it works.

In the pgWelcome window, select the Phone Layout tab at the bottom. You should see a blank phone:

On the Ribbon Bar, select the Layout tab so that the Ribbon Bar now looks like this:



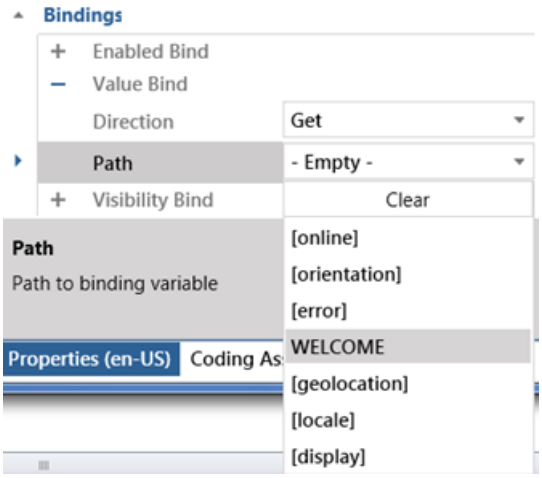
We are going to add a Text field – it's the first in the Control section, with the "A" icon. You can click-and-drag it down to the phone layout, or just double-click it. The text field will appear like this:



Notice that when you are in Phone Layout mode, the window on the right side of TD Mobile, by default, is the Properties Window. (If for any reason that is not showing, click the Properties icon on the Layout tab of the Ribbon

Bar.) If we wanted to assign the text field a value at design-time, as you would for a label, say, we could set its Caption property; but we want to demonstrate how we would get data from server-side code, at run-time, so we are going to set its Value Bind property.

With the text field selected, look down near the bottom of the Properties window for the **Bindings** section. Click on the plus sign ("+") next to the **Value Bind** item; then in the **Path** dropdown box, click on the down arrow and select the Binding we declared as 'WELCOME'. (The other options you see there are special System Bindings that you will learn about later).

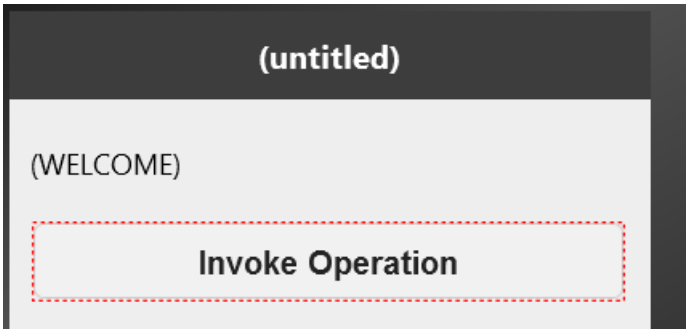


If successfully set, the text field will now show the name of its Value Bind while in design-mode:

Now let's add a Button. It's the second Control icon from the left in the top row in the Controls section of the Layout tab on the Ribbon Bar. (If you let the mouse hover over the icons, you'll see the control name in a tooltip.)

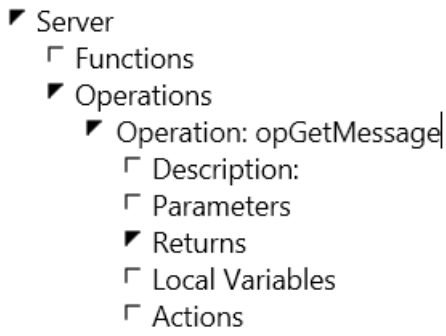
Double-click the icon to add the Button to the page. Change the Button's Caption, in the Properties window, to "Invoke Operation".

Your phone layout should now look like this:



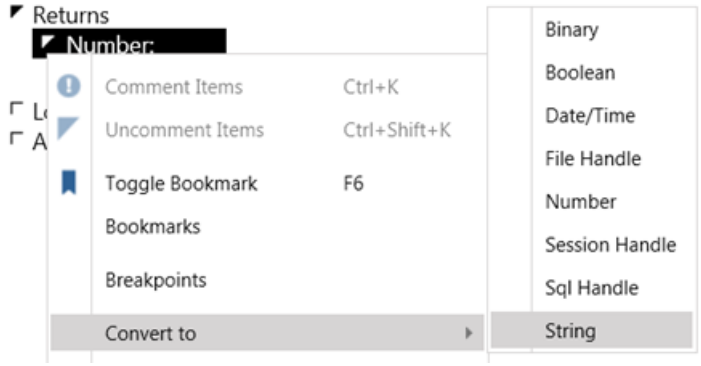
The next thing we'll add is the Operation that we will invoke. An Operation is a server-side function, but a special function that only client code can call. It is the server's access to the Bindings you declare on your client-side web page. In the pgWelcome window, click on the Outline tab at the bottom.

Double-click the Server node at the bottom of the Outline, to expand it, then double-click the Operations node and from the dropdown list of options, choose Operation. Next to the Operation node, type in a name for it, let's use opGetMessage. Our Server section should now look like this:

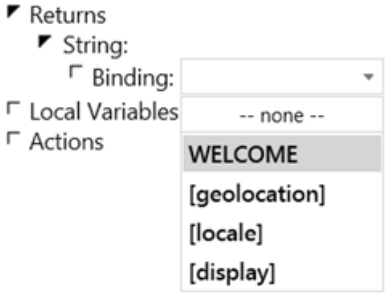


Let's define the Returns section for our operation. Double-click the Returns node, and then double-click the Number node, so that it now looks like this:

First we must change that default data type from Number to String. You could simply select the text of 'Number' and type over it, or to avoid typos, you can right-click the Number node, select **Convert To**, then select **String**:



Now that we have the data type we need, we tie it to our Binding. Click on the Binding's down arrow and select WELCOME:



(The Binding dropdown, by the way, will only show Bindings of the data type specified; if we had left it as Number, 'WELCOME' would not display.)

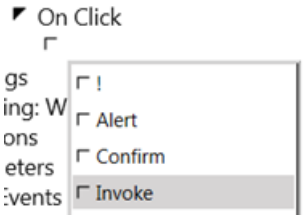
Now your Operation's Return section should look like this:

Now, in the Actions section, which is where your code goes, we will add a single line of code. We'll use the Return statement with a string literal, to populate our client's Binding with a value. Here's the entire Operation:

So to recap the wiring we've done here:

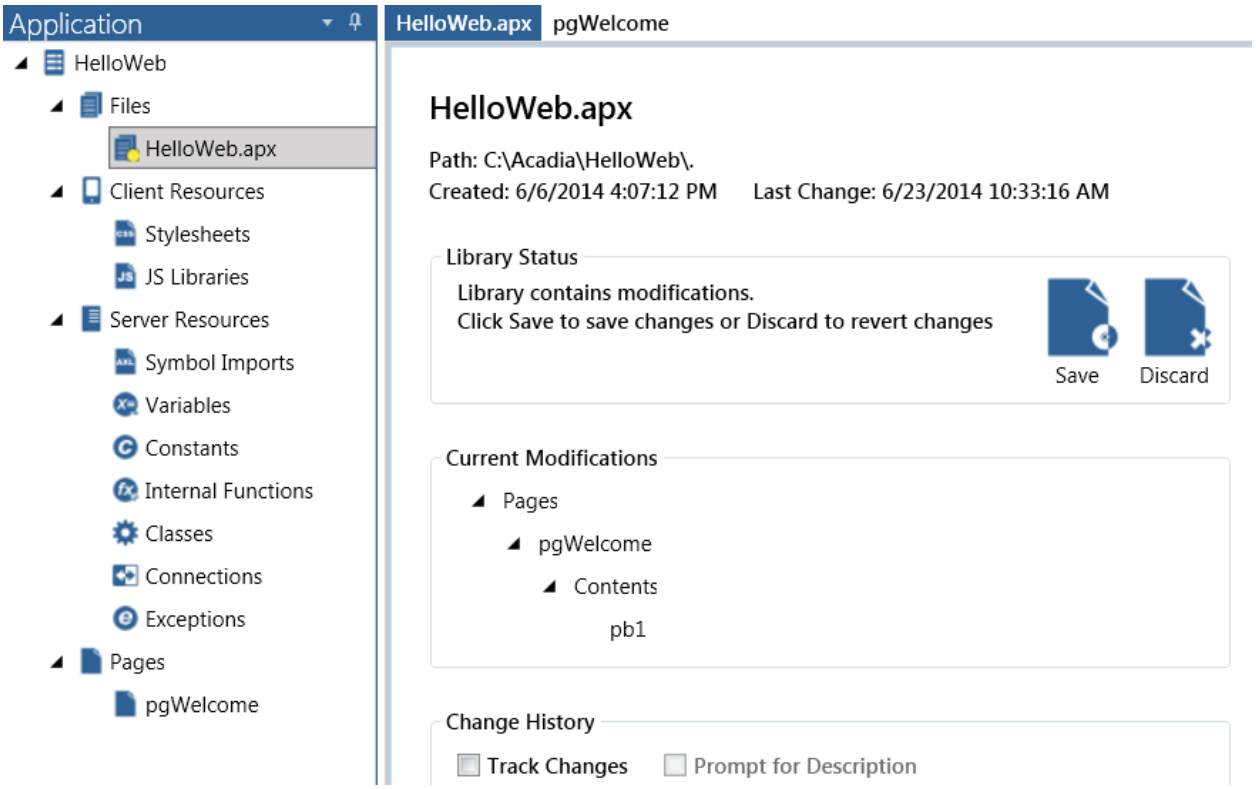
1) First we declared a Binding of type String
2) Then we set the Value Bind of our text field to that Binding
3) Then we specified that Binding as the Return value of an Operation

Now we just need to call, or as we say, *invoke* our Operation. We're going to invoke it when the user clicks the button on our phone, so we need to add an Invoke command in the button's On Click event. You can navigate to this through the Outline window, but an easy way to do it is to go to the Phone Layout window where we set the Caption of our button to 'Invoke Operation', - and double-click the button. That will insert an On Click event for us (if one doesn't already exist) and re-open the Outline window right at the event. Double-click the On Click node and select Invoke from the available command options:



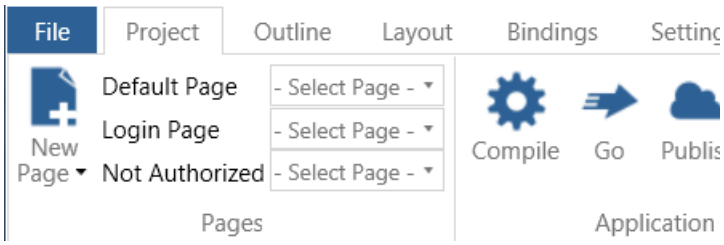
Choose our Operation from the dropdown list of available Operations, and we are ready to go:

Let's save our work to this point. There are a number of ways to do this; Control + S will work. But this is a good opportunity to check out the file page. In the Application window, within the Files section, click on the file name: HelloWeb.apx. Here you can not only Save or Discard all pending changes, but you can see a list of those changes, and you can select options for how you record your Change History:

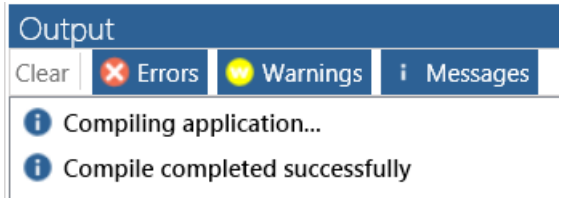


Before we run the app, let's look at what we have in our HelloWeb folder – nothing but a tiny 3KB file named HelloWeb.apx (unless you have already run or compiled your app). Open the .apx with Notepad or some other text editor, if you are curious, and you'll see that it is a simple text file, in xml format.

Now go back to TD Mobile and compile the app. The **Compile** button is on the Project tab of the Ribbon Bar, in the Application section. It has an image like a gear as an icon:

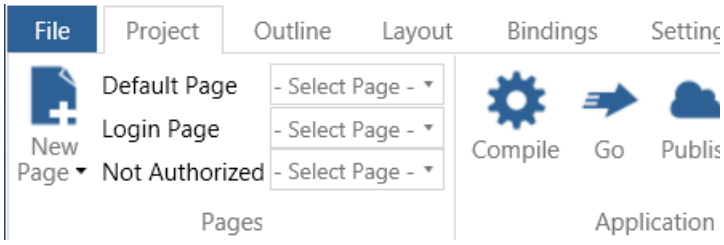


If all is well you will see this result in the Output window at the bottom of TD Mobile (if not the compiler should point you to the problem):

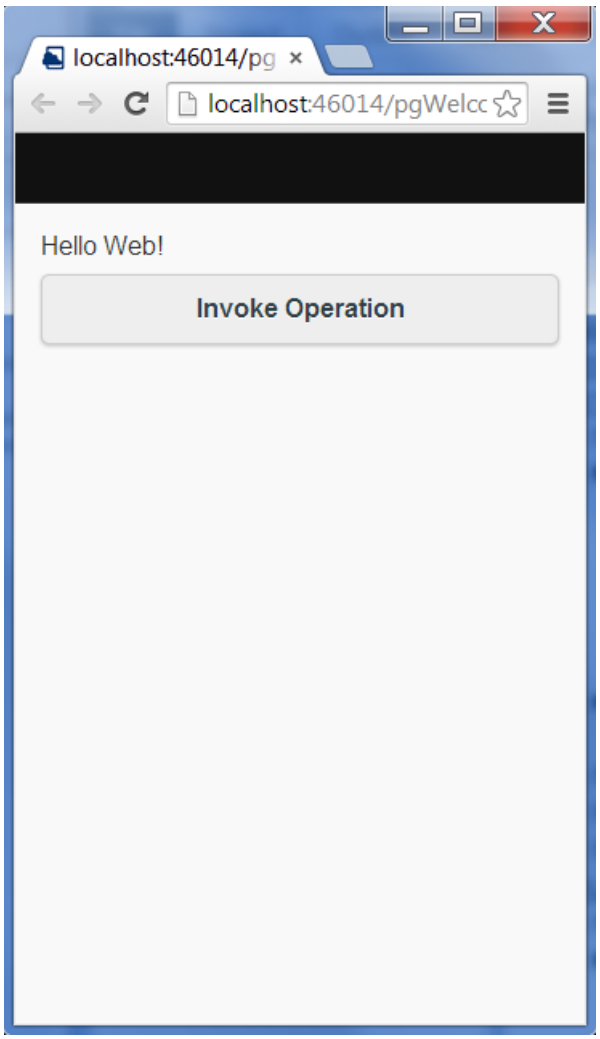


Now go back to the HelloWeb folder and see how much has been added: html, javascript, css files, etc. – everything you need for your web app.

Now let's run it. You can use the shortcut key F7 or click the Go button. In the Project tab of the Ribbon Bar, the **Go** button is next to the Compile button, with an arrow icon:

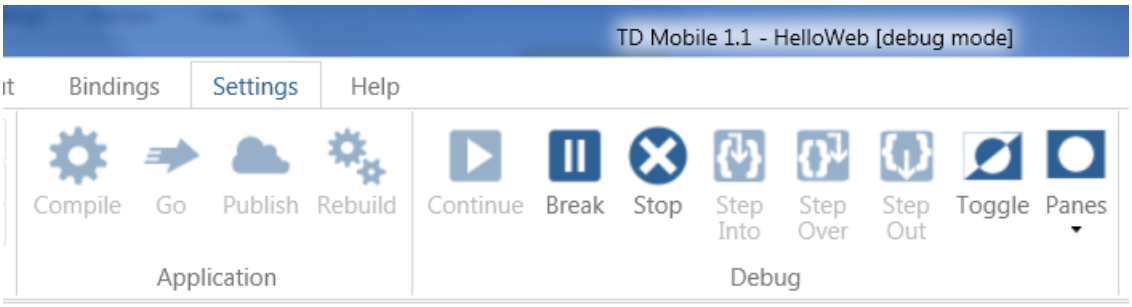


When your browser runs, and displays your phone layout, click the button and you should see our enthusiastic message:



When running your app on your local development machine, TD Mobile runs IIS Express to locally host your app and uses your default web browser to display your web page.

If you close the browser to close the app, note that TD Mobile will still be running. You will need to click the Stop button (in the Project tab of the Ribbon Bar, in the Debug section), or use the shortcut keys (Shift + F7), to take TD Mobile out of debug mode.



So that's a simple example of using a Binding. Maybe a little silly, but you'll see it's basically the same process to get a list of customers from a database. Bindings can also be arrays and user-defined types (Classes) and even arrays of user-defined types.

We used a Binding as an Operation's Return value, as a way to get data *from* the server. If you want to pass data *to* the server, define Parameters for the Operation, making sure to specify a Binding for each one.

See more examples of using Bindings with different types of controls in the document *TDMobile_Primer v1.pdf*, which you can find in the whitepapers folder within your TD Mobile installation folder.

Deployment

Let's pretend you wanted to deploy your HelloWeb app to a web server so that you could access it with a mobile device from the internet.

Of course you need to have a web server set up. On your web server, IIS7 or higher and .NET4 (currently) are required. The version of Windows should be Windows 2008 or higher (server) or Vista or higher (workstation). Run the file *tdmdeployer_xxbit.exe* to install the TD Mobile runtime files and other necessary files; you can find it in the deployer folder in your TD Mobile installation folder.

Here's one way to deploy your app. Copy the entire HelloWeb folder from your development machine, and paste it into the IIS applications folder (by default it is at C:\inetpub\wwwroot) of your web server. Then run the IIS Manager on the web server, and in the Connections window, dig down into the tree view until you get to Default Web Site; then find the HelloWeb folder. Right-click on that folder and select Convert to Application. Make sure the Application Pool is "ASP .NET v4.0" (this will likely change in the future, see "Setting up a TD Mobile application in IIS" in the TD Mobile Help). But that's basically it. You could access the default page in your app with any browser by navigating to: http://<YourWebServersUrl>/HelloWeb.

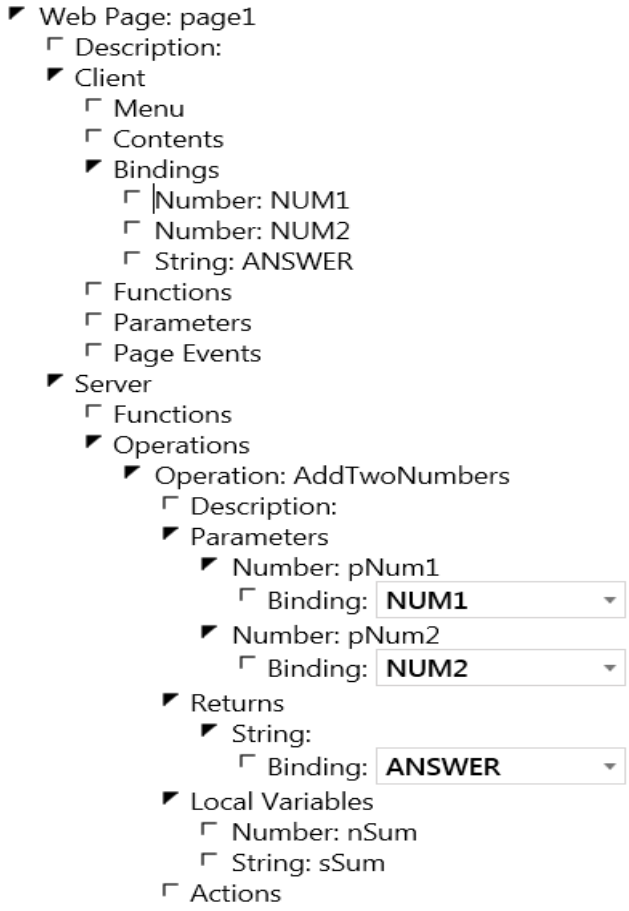
TD Mobile offers more sophisticated ways of doing this via the Publishing functionality. Read more about that in the document *TDMDeployer.pdf* in the whitepapers folder of your TD Mobile installation folder.

What is SAL?

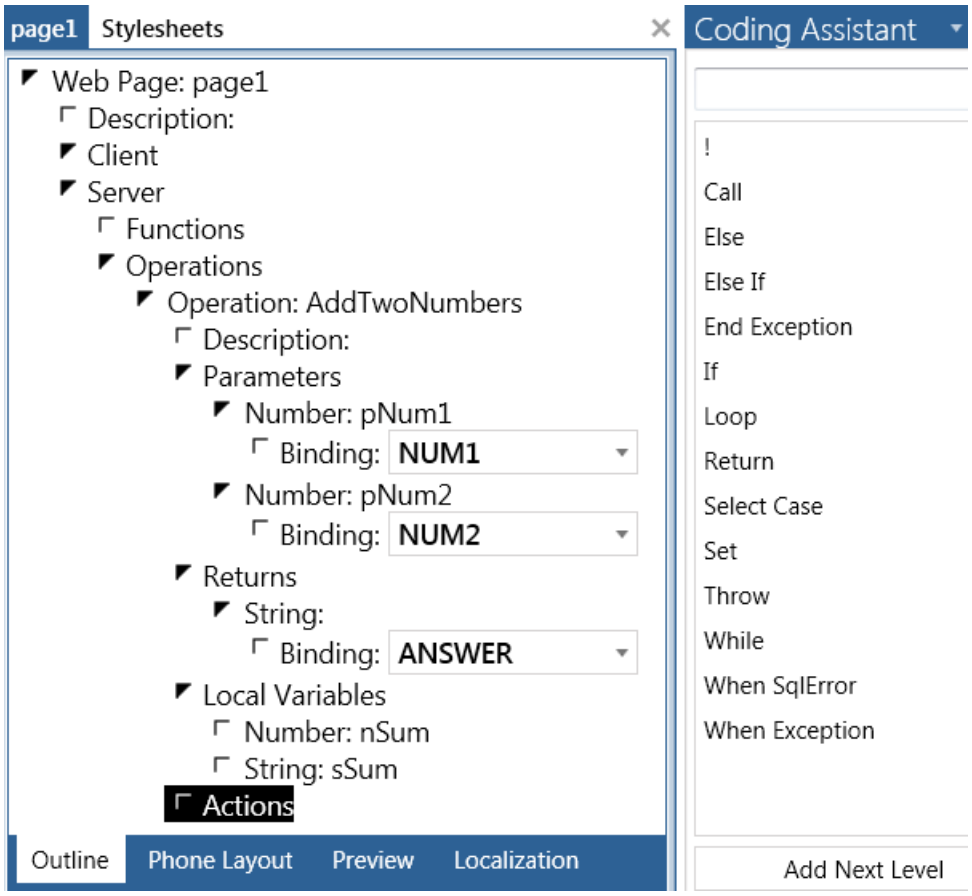
SAL is the server-side application language for TD Mobile. SAL is a powerful .NET language that you use to build server side logic. SAL is a complete programming language that lets you do any kind of computations and interface with the power of the Microsoft .NET architecture. SAL allows you to call Web Services; you can call all .NET framework methods for example for file access or to programmatically post a web form to a server just to mention a few of the capabilities. SAL has a large number of built-in functions that provide SQL database access, email sending, XML access and much more.

Before we take a look at the nuts and bolts of the language in the next section, *SAL Components*, let's take a look at how to get started writing code in TD Mobile, and particularly how to use the Coding Assistant and the built-in intellisense. For a simple example, say we want to code a server-side operation to receive two numbers from the client and then return the answer in a text message.

First we need to wire up three Bindings. Our Operation will have two Number parameters and return a String:

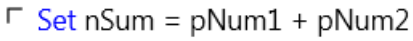


The code goes into the **Actions** section of our Operation. Select that Actions node and then, in the Coding Assistant, we see all the available SAL commands available for us to begin our first line of code:

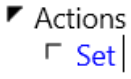


Our first line of code will add the two numbers together and assign the total to our variable nSum. To do that we need the **Set** command, the assignment command. From the Coding Assistant list, double-click "Set". A line of code is inserted with the selected command followed by the cursor, ready for us to finish the line of code:

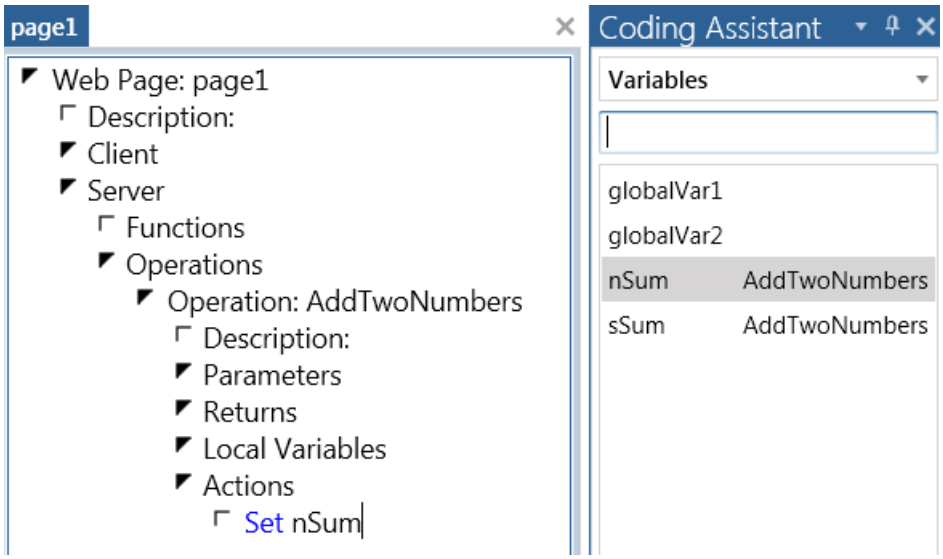
Normally, you might have the variable and parameters fresh in mind and you would just type in the line we need, specifying the variable, the equal sign (the assignment *operator* in TD Mobile), and the expression adding the two parameters:



However I want to show you other possibilities, so let's go back to this point:



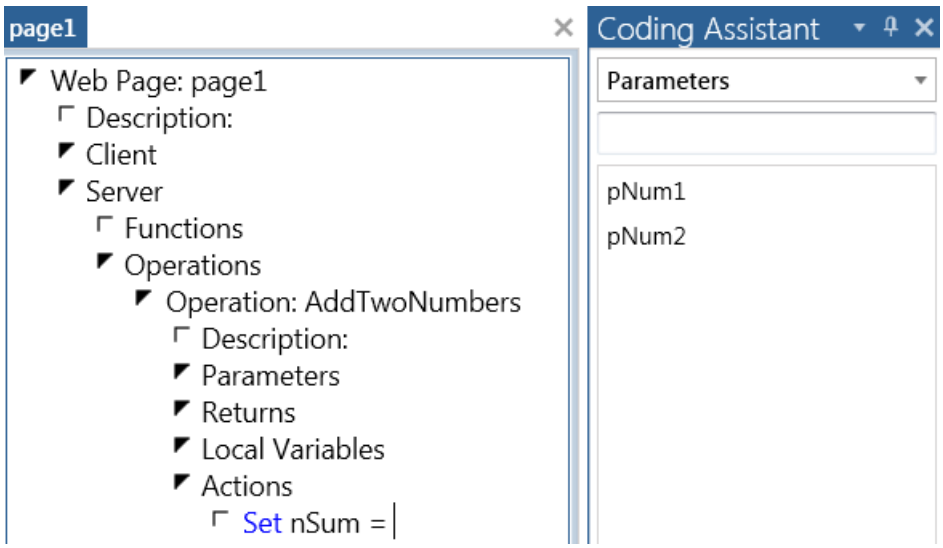
Pretend we have a large amount of variables in this Operation and, especially with the large Variables section closed to save screen space, we're staring at that Set with the cursor blinking and we're wondering "What did we name that variable?" When in doubt ask the Coding Assistant. Select **Variables** from the dropdown box at the top of the Coding Assistant to see all the variables available in our current scope. (I added two global variables for demonstration purposes. Notice that the local variables are identified by the operation or function that defines their scope, while the global variables have no such limitation.) Double-click **nSum** and it will be inserted into your code line:



Now type in an equal sign ("="), the assignment operator in TD Mobile:

Set nSum =

Again your memory needs a nudge, "What were those parameter names?" Select Parameters in the Coding Assistant dropdown:



Select pNum1, then type the addition operator ("+") and then select pNum1. Now the line is finished:

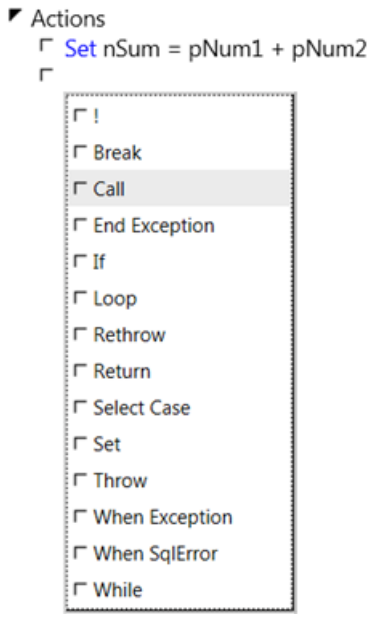
Set nSum = pNum1 + pNum2

This may or may not suit your coding style, but it would absolutely cut down on typos.

For our next line of code we want to call a function that will convert the number held by **nSum** into a string, and assign that string to our other local variable **sSum**. Hit the Enter key or otherwise select the first line:

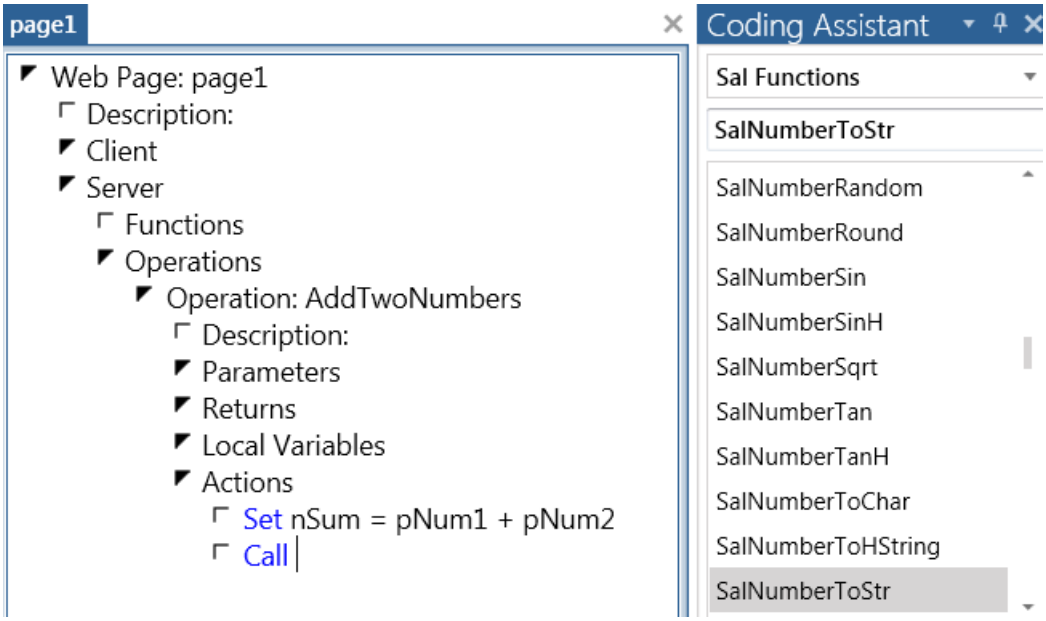
Actions
Set nSum = pNum1 + pNum2

Then hit the Insert key to insert the next line, and you get another look at the available commands:



You can double-click your choice from the list or just ignore it and type. This time just type "Call " and that list will go away. TD Mobile strives to walk the fine line of being a helpful guide *and* knowing when to get the heck out of the way.

In the dropdown list at the top of the Coding Assistant, choose Sal Functions. In the text field below that dropdown, type "SalNumber". The list of functions will scroll you down to where all the SalNumber functions are. Possibly you will be able to guess by the function's name which one will do what you need, or you may have to go into the SAL Help (on the Ribbon Bar, Help tab, the SAL icon is 2nd from the left) and see the documentation for some of these functions. Often the function name will be self-explanatory. Scroll down the list, or type an experienced guess into the text field – to get to SalNumberToStr; that's the function we need:



Select SalNumberToStr in the Coding Assistant list and it will be inserted into the line of code with a list of the arguments it takes, displaying the data types of those arguments, highlighting the first for you to replace with a variable or a literal value or an expression:

```
Set sSum = SalNumberToStr(Number, Number, String)
```

If this is your first time using this function, you're going to need more help than this. You could search for the function in the SAL Help and get the full documentation for this function. Often, though, intellisense will be all the help you need. The intellisense is triggered when you type the opening parenthesis after the function name. So let's delete everything after the function name and re-type that parenthesis. Then the intellisense for the function will come up:

```
Call SalNumberToStr(
```

```
Number = SalNumberToStr( Number, Number, String )
 Converts a number to a string.
```

First you see the basic description of the Function: "Converts a number to a string."

Now to learn about the first argument, hit the space bar to move the cursor one character to the right, and the intellisense changes:

```
Call SalNumberToStr(
```

```
Number = SalNumberToStr( Number, Number, String )
 Converts a number to a string.
 nNumber: Number. The number to convert
```

Ah, "the number to convert". Type in the variable holding the number we want to convert, *nSum*, and a comma, and then we see info about the next arg:

```
Call SalNumberToStr( nSum,
```

```
Number = SalNumberToStr( Number, Number, String )
 Converts a number to a string.
 nDecimalPlaces: Number. The number of decimal places
 you want in strString.
```

The number of decimals we want: let's say 2, then type a comma to see info about the 3rd arg:

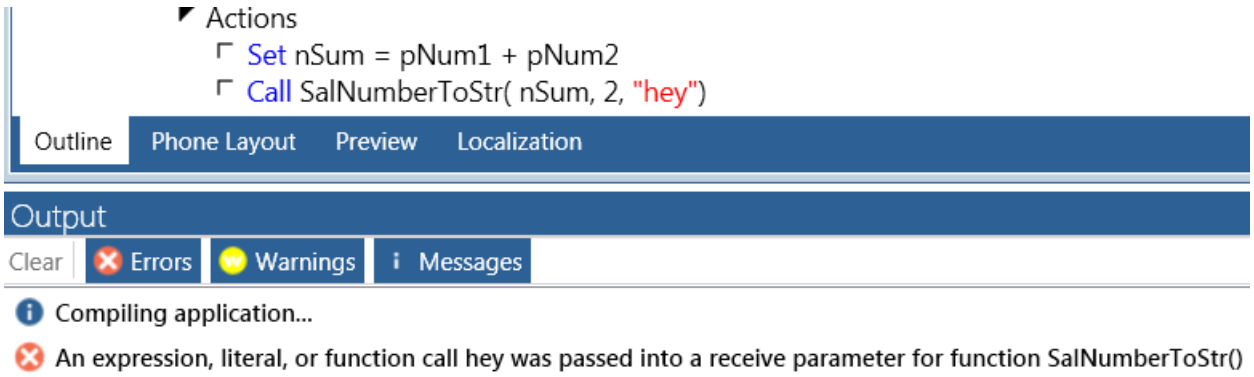
```
Call SalNumberToStr( nSum, 2,
```

```
Number = SalNumberToStr( Number, Number, String )
 Converts a number to a string.
 strString: Receive String. The string converted from
 nNumber.
```

Then type in the string variable that will hold the converted-to-string number, *sSum*, (or use the Coding Assistant to find it, if you like), and type the closing parenthesis:

```
Actions
  Set nSum = pNum1 + pNum2
  Call SalNumberToStr( nSum, 2, sSum )
```

Note that the third argument was described as a *Receive String*, meaning that the function will change the variable and return it with a new value. Such arguments must be a variable. If you tried to pass the function a non-variable, at compile time you would learn that this is not allowed:



The compiler is another kind of coding assistant, and will help you write valid SAL code. If you click on that line marked with the red "X" icon, the offending line of code will be displayed and selected.

To finish up our operation we will insert a Return statement and concatenate a string literal and a variable that will populate the Binding *ANSWER* back on the client.

```
Actions
  Set nSum = pNum1 + pNum2
  Call SalNumberToStr( nSum, 2, sSum )
  Return "The answer is: " || sSum
```

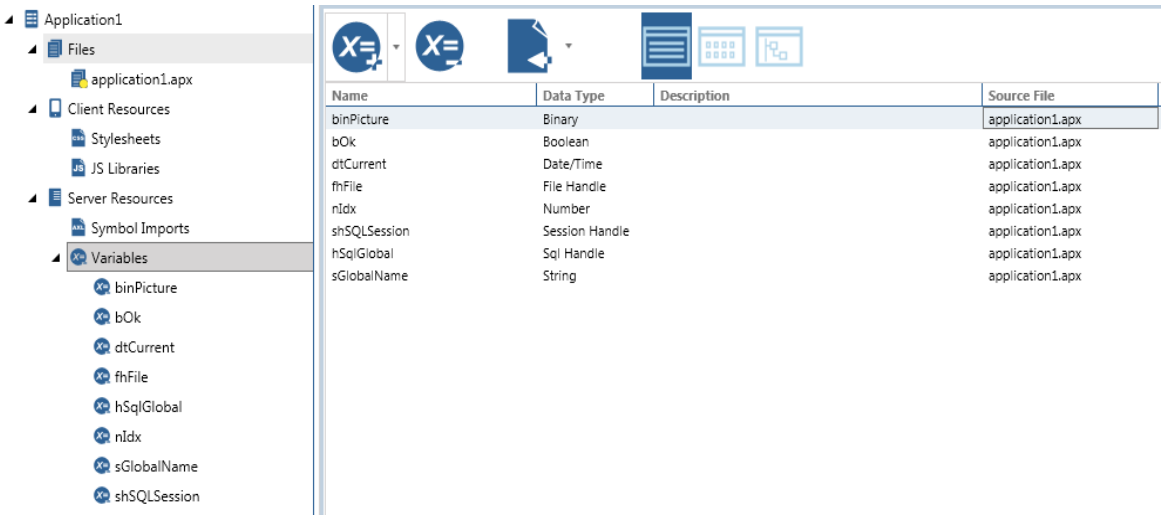
SAL Components

Data types

You specify a data type for variables and constants. Variables can be one of these data types:

- Binary
- Boolean
- Date/Time
- File Handle
- Number
- Sql Handle
- SessionHandle
- String

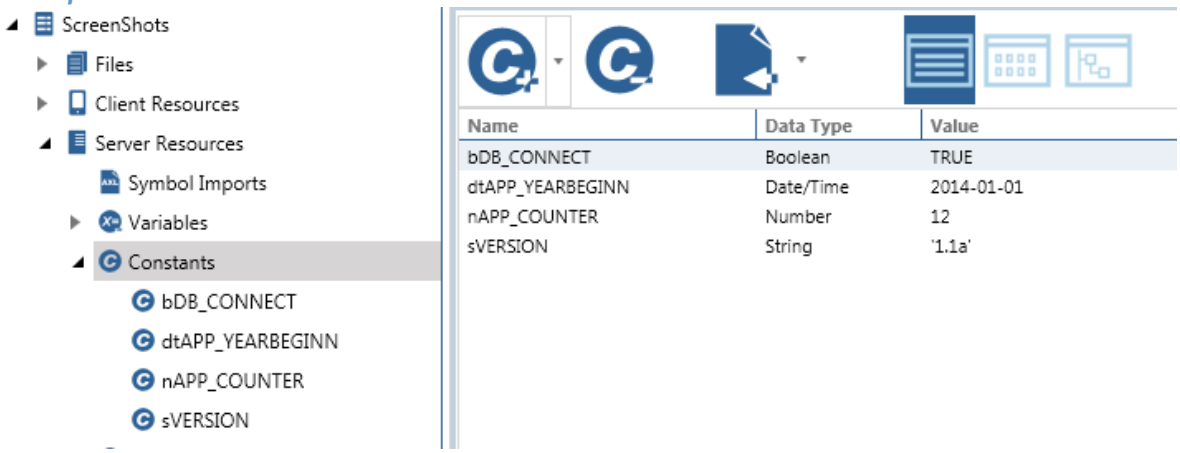
Example:



Constants can be one of these data types:

- Boolean
- Date/Time
- Number
- String

Example:



Receive data types

All data types can be an alternate form called a *receive* data type. Receive parameter are used in parameter lists of global and local functions.

Example:

- Function: fnCalculateDateNumbers
  - Description:
  - Returns
    - Boolean:
  - Parameters
    - Date/Time: dtIn
    - Receive Number: nYear
    - Receive Number: nMonth
    - Receive Number: nDay

Binary

This data type, introduced in TD 5.2, provides better support for binary data within SAL, including data from BLOB columns.

To assign a value to a BINARY variable, use SalPicSetBinary, the BINARY_Null constant, or another BINARY variable.

Example:

- Function: fnCalculateDateNumbers
  - Description:
  - Returns
  - Parameters
  - Static Variables
  - Local Variables
    - Binary: binPicture

Boolean

Use this data type for variables that can be TRUE or FALSE. These values are system constants: TRUE is 1 and FALSE is 0.

Example:

- Function: fnCalculateDateNumbers
  - Description:
  - Returns
  - Parameters
  - Static Variables
  - Local Variables
    - Boolean: bOk
  - Actions
    - Set bOk = FALSE

Date/Time

Use this data type for dates and times. The default output format is ISO:

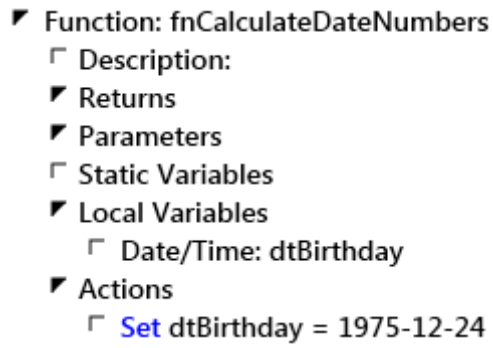
YYYY-MM-DD-HH.MM.SS.MSMSMS

The only valid input format for Date/Time values in Set statements is ISO as shown above. Note the following:

- The year must be four digits
- The month, day, hour, minute, and seconds must be 2 digits. Include a leading zero when the value is less than 10
- You must use the hyphens and periods as separators in the positions shown above
- The microseconds (MS) can be up to six digits

You can use the DATETIME_Null system constant to set a Date/Time to a null value, or to check if a Date/Time value is null.

Example:



Internally, TD Mobile stores Date/Time data in its own floating point format. This format interprets a Date/Time value as a number in this form:

DAY is a whole number that represents the number of days since December 30, 1899. December 30, 1899 is 0, December 31, 1899 is 1, and so on.

TIME is the fractional part of the day. Zero represents 12:00 AM, .25 is 6:00 AM, .5 is 12:00, .75 is 3:00, and so on.

For example, March 1, 1900 12:00:00 PM is represented by the floating value 61.5 and March 1, 1900 12:00:00 AM is 61.0.

If you omit a part of an input Date/Time value, TD Mobile supplies the default of 0, which converts to December 30, 1899 (date part) 12:00:00 AM (time part).

For example, if you define this variable:

```
Date/Time: dtExample
```

and execute this Set statement that does not specify a time:

```
Set dtExample = 1983-10-02
```

then the value in dtExample is:

```
1983-10-02-00.00.00
```

Note: When the microseconds part is zero, TD Mobile omits the microseconds in its default output format.

Date/Time arithmetic

You can perform these arithmetic operations with Date/Time values:

- Add a Number value to a Date/Time value, giving you a Date/Time value
- Subtract a Number value from a Date/Time value, giving you a Date/Time value
- Subtract one Date/Time value from another Date/Time value, giving you a Number value

Note that if you add or subtract a Number value to or from a Date/Time value, the result is a Date/Time value.

The next sections show examples of each type of Date/Time arithmetic. In these examples, these variables are used:

```
Date/Time: dtExample1
Date/Time: dtExample2
Number: nResult
```

Adding a Number to a Date/Time

When you add an integer to a Date/Time, TD Mobile adds that many days to the value. If you execute these statements:

```
Set dtExample1 = 1983-10-02
Set dtExample2 = dtExample1 + 32
```

Then the result in dtExample2 is:

```
1983-11-03-00.00.00
```

Subtracting a Number from a Date/Time

When you subtract an integer from a Date/Time, TD Mobile subtracts that many days from the value. If you execute these statements:

```
Set dtExample1 = 1983-10-02
Set dtExample2 = dtExample1 - 32
```

Then the result in dtExample2 is:

```
1983-08-31-00.00.00
```

Subtracting a Date/Time from a Date/Time

When you subtract a Date/Time from another Date/Time, TD Mobile finds the number of days between the two dates. If you execute these statements:

```
Set dtExample1 = 1986-01-12
Set dtExample2 = 1983-10-02
Set nResult = dtExample1 - dtExample2
```

Then the result in nResult is:

```
833
```

Using decimal numbers in Date/Time arithmetic

TD Mobile treats the digits to the right of the decimal as the percentage of a day. If you execute these statements:

```
Set dtExample1 = 1986-01-12
Set dtExample2 = dtExample1 + .25
```

Then the result in dtExample2 is:

```
1986-01-12.06.00.00
```

If you execute these statements:

```
Set dtExample1 = 1986-01-12
Set dtExample2 = dtExample1 + .99999
```

Then the result in dtExample2 is:

```
1986-01-12.23.59.59.136000
```

Year 2000 support

TD Mobile determines the value for a user's 2-digit century entry as follows:

1. Assume the current year is 1996:

```
If 05 is entered, the computed date is 2005
```

```
If 89 is entered, the computed date is 1989
```

2. Assume the current year is 2014:

```
If 05 is entered, the computed date is 2005
If 34 is entered, the computed date is 2034
If 97 is entered, the computed date is 1997
```

3. Assume the current year is 2065:

```
If 05 is entered, the computed date is 2105
If 70 is entered, the computed date is 2070
```

Number

Use this data type for numbers with up to 44 digits of precision. You can use the NUMBER_Null system constant to set a Number to a null value, or to check if a Number value is a null.

If you use a Number data type as a bind variable to write a SQLBase DECIMAL data type column, truncation can happen because SQLBase DECIMAL data types have a maximum of 22 digits of precision.

Example:

```
Local Variables
    Number: nMonth
Actions
    Set nMonth = SalDateMonth( SalDateCurrent( ) )
```

Sql Handle

Use this data type to identify an open connection to a database. All access to a database requires a Sql handle. You use Sql Handles in Sql* functions to execute SQL statements.

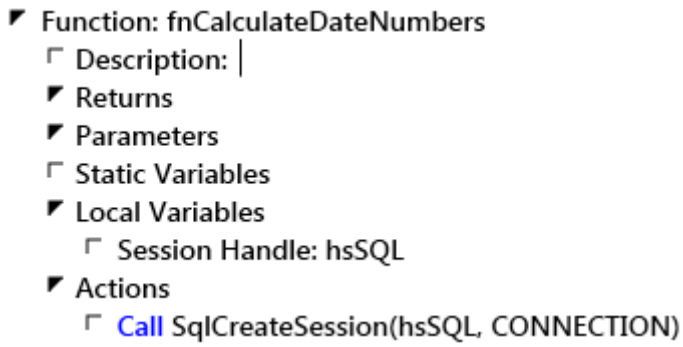
Example:

SqlConnect returns the handle. Before you call SqlConnect, hSql does not have a valid value.

```
Local Variables
    Number: nE_ID
    Number: nRet
    Sql Handle: hSQL
Actions
    If SqlConnect(hSQL)
        If SqlPrepareAndExecute(hSQL, 'select E_ID from Employee into :nE_ID where Status = 1')
            Call SqlFetchNext(hSQL, nRet)
        Call SqlDisconnect(hSQL)
```

Session Handle

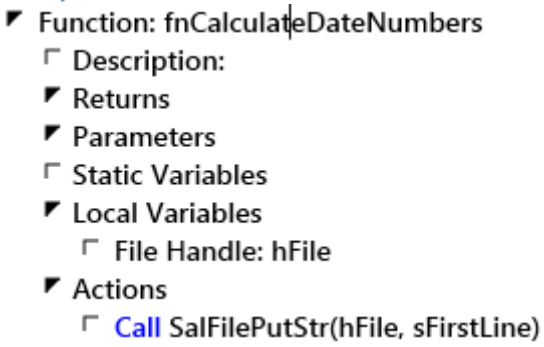
You use this data type for multi-connection transactions and OLE DB provider Connections.



File Handle

Use this data type to identify an open file. When you open or create a file, TD Mobile returns a file handle. You then use the file handle to identify the file.

Example:



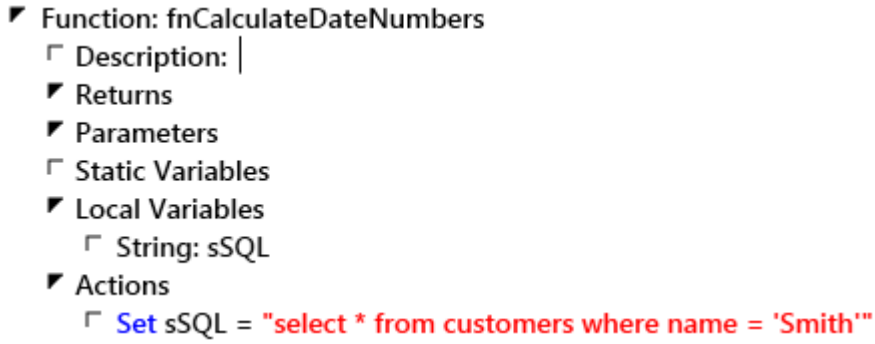
String

Use this data type for character data. The only limit on the length of a String data type is available system memory.

Enclose literal strings in single quotes. You can also enclose literal Strings in double quotes. When you do, you do not need to put escape characters before embedded single quote characters. For example:

```
String: strSelect = "select * from customers where name = 'Smith'"
```

Example:



Data types treated as Booleans

Strings, numbers, dates, and handles (file and SQL), are automatically converted (“cast”) to a Boolean when used as an operand of an “If”, “While”, or “Enabled when” statement, or used as an operand of an “AND”, “OR”, or “NOT” operator.

An uninitialized variable, of any data type, when converted to a Boolean, evaluates to FALSE.

A variable, of any data type, which has been assigned a null value from a database, evaluates to FALSE.

A string variable or constant with the value '' (null string) evaluates to FALSE. A number variable or constant with the value 0 (zero) evaluates to FALSE.

Everything else evaluates to TRUE.

Variables

A variable can hold any value of its data type.

Where you declare variables

You define variables in these places:

- Server Resources (Variables section)
- Class Functions, Internal (Global) Functions and Operations (Parameters, Static Variables, and Local Variables sections)
- Class Definitions (Instance Variables sections)

Syntax

Use this syntax to declare a variable:

```
Data Type: VariableName
```

These are examples of variable declarations:

```
Boolean: bReturn
Date/Time: dtBirthday
Number: nCount
Sql Handle: hSql
String: strName
```

When variables are valid

Variables in the Server Resources section are valid as soon as the application starts. You can refer to global variables in any SAL statement.

Variables in the Local Variables section of a function definition are valid when you call the function and become invalid when the function returns.

Variables in SQL statements

You use variables in SQL statements in two ways:

- To bind input data to a SQL statement. Variables used in this way are called *bind variables.*
- To specify where to put the output of a SQL SELECT statement. The INTO clause specifies the variables where query data is placed. Variables in an INTO clause are called *into variables*. When you use variables in a SQL statement, you must prefix them with a colon (:).

Variable Types: C# vs. TD Mobile

C#	Represents	Range	TD Mobile
bool	Boolean value	True or False	Boolean
byte	8-bit unsigned integer	0 to 255	Number
decimal	128-bit precise decimal values with 28-29 significant digits	$(-7.9 \times 10^{28}$ to $7.9 \times 10^{28}) / 10^{0 \text{ to } 28}$	Number
double	64-bit double-precision floating point type	$(+/-)5.0 \times 10^{-324}$ to $(+/-)1.7 \times 10^{308}$	Number
float	32-bit single-precision floating point type	-3.4×10^{38} to $+ 3.4 \times 10^{38}$	Number
int	32-bit signed integer type	-2,147,483,648 to 2,147,483,647	Number
long	64-bit signed integer type	-923,372,036,854,775,808 to 9,223,372,036,854,775,807	Number
sbyte	8-bit signed integer type	-128 to 127	Number
short	16-bit signed integer type	-32,768 to 32,767	Number
uint	32-bit unsigned integer type	0 to 4,294,967,295	Number
ulong	64-bit unsigned integer type	0 to 18,446,744,073,709,551,615	Number
ushort	16-bit unsigned integer type	0 to 65,535	Number
String			String
char	16-bit Unicode character	U +0000 to U +ffff	String

Arrays

An array is a collection of variables (elements) of the same data type that you refer to with a common name. You refer to an individual element in an array with a number that represents the index offset.

An array can be static or dynamic:

- A static array contains a fixed number of elements
- A dynamic array contains a variable number of elements

An array can be one-dimensional or multi-dimensional (an array of arrays).

TD Mobile always passes array elements to functions by reference even if the function parameter is declared with the Receive keyword.

One-dimensional arrays

Static arrays

If you know the maximum number of elements that an array can contain at one time, specify that number when you declare the array:

```
String: strEmployees[10]
```

The ten elements in the array above are numbered 0-9. An array like this with a fixed number of elements is called a static array. You must specify a numeric literal for the number of elements.

You can put any expression that evaluates to a number between the square brackets.

Dynamic arrays

If you cannot predict the maximum number of elements in an array, use an asterisk instead of a number to tell TD Mobile that it is a dynamic array:

```
String: strEmployees[*]
```

The elements in the array above are numbered 0-n, where n depends on available system resources.

Dynamic arrays initially have zero elements. Call SalArrayIsEmpty to determine if an array contains data. You can reset a dynamic array to zero elements by calling SalArraySetUpperBound and setting the nBound parameter to -1.

Setting array bounds

By default, you refer to the first element of an array with zero. To control how you refer to the elements in an array, specify the lower bound (or lower "range") and the upper bound (or upper "range"). Separate the two numbers with a colon:

```
String: strEmployees[1:10]
```

The ten elements in the array above are numbered 1-10.

You can set the lower bound in a dynamic array:

```
String: strEmployees[1:*]
```

The elements in the array above are numbered 1-n, where n depends on available system resources. Important: You cannot specify an asterisk for the lower bound.

Referring to arrays

You refer to an element in an array by specifying its index:

```
Set df1 = strEmployees[5]
```

The index can be any expression that evaluates to a number.

Multi-dimensional arrays

You declare a multi-dimensional array like a one-dimensional array, but you also specify the number of elements in the second and subsequent dimensions after the number of elements in the first dimension. You separate each dimension specification with a comma.

Note: The maximum number of dimensions in an array is limited only by available system resources.

Static arrays

This example declares a 2-dimensional array with a fixed number of elements in both dimensions:

```
String: strEmployees[10, 3]
```

The array above has ten elements in its first dimension (numbered 0-9) and three in its second dimension (numbered 0-2).

Dynamic arrays

You can make the first dimension dynamic:

```
String: strEmployees[*, 3]
```

The array above has a dynamic number of elements in its first dimension (numbered 0-n) and three in its second dimension (numbered 0-2).

Important: You can make only the first dimension of a multi-dimensional array dynamic.

Setting array bounds

You can control how you address the elements in any dimension:

```
String: strEmployees[1:10, 1:3]
```

The array above has ten elements in its first dimension (numbered 1-10) and three in its second dimension (numbered 1-3).

You can set the lower bound if the first dimension is dynamic:

```
String: strEmployees[1:*, 1:3]
```

The array above has a dynamic number of elements in its first dimension (numbered 1-n) and three in its second dimension (numbered 1-3).

Referring to multi-dimensional arrays

You refer to elements in a multi-dimensional array the same as you would in a one dimensional array. The difference is that for a multi-dimensional array you specify the second and subsequent dimensions' index after the first dimension's index. You separate each index with a comma. For example:

```
Set df1 = strEmployees[2, 5]
```

Constants

A constant contains a single, unchanging value. You can declare a constant as one of these data types:

- Boolean
- Date/Time
- Number
- String

You can only declare constants in the Constants section within the Server Resources section in the Application window. You can refer to a constant wherever you can refer to a variable.

You can declare numeric constants with hexadecimal values. For example:

```
0x1234ABCD
```

Syntax

Use this syntax to declare a constant:

```
Data Type: ConstantName = expression
```

Examples:

```
Constants
    Number: BASE = 500
    Number: MAXNUM = BASE+1000
    String: STATE = 'New Jersey'
    String: City = 'Newark'
    String: PLACE = CITY || ',' || STATE
    Date/Time: July_4 = 1994-07-04
    Boolean: bDone = FALSE
```

Naming conventions

Variables

Use prefixes in the names of variables to make the outline self-documenting. The table below lists the name prefixes.

Data type	Name prefix	Example
Boolean	b	bVarName
Date/Time	dt	dtVarName
File Handle	fh	fhFileVarName
Number	n	nVarName
Sql Handle	hSql	hSqlVarName
String	s (or) str	sVarName

Constants

Use an uppercase prefix with an underscore followed by a mixed-case or uppercase name. For example:

```
TYPE_ConstantName
TYPE_CONSTANTNAME
```

Operators

An operator is a symbol or word that represents an operation to perform on one or more values. The table below shows the operators:

Operator symbols	Operator type
+, -, *, /	Numeric
unary -	Unary
=, !=, >, <, >=, <=	Relational
AND, OR, NOT	Boolean
&	Bitwise AND
|	Bitwise OR
||	Concatenate String

Expressions

An expression is a combination of constants, variables, and operators that yields a single value. An expression can be:

- The result of a function
- A variable
- A constant
- Two or more expressions connected with an operator

TD Mobile uses these precedence rules to evaluate expressions:

- Evaluate expressions with AND, OR, and NOT from left to right
- Stop evaluating AND/OR as soon as the result is known
- Evaluate expressions in parentheses first

Examples:

```
nSalary[grade] + .1*nSal[3]
bQueryOn
MAXNO
1 + 1
SalDateCurrent( )
```

Control Structures

If – Else If – Else

Use If, Else or Else If to express options. Indentation determines the conditional flow of control.

The Else If or Else portion is optional. You can add as many levels of Else If statements as you like, but there can only be one Else statement.

The syntax is:

```
If Expression1
        Statement1
Else If Expression2
        Statement2
Else
        Statement3
```

In the syntactic example above, TD Mobile evaluates Expression1. If it is true, Statement1 executes. If it is false, TD Mobile evaluates Expression2. If Expression2 is true, Statement2 executes. If it is false, Statement3 executes.

Example:

```
If nCountryID = 49
   Set nTax = 19
Else If nCountryID = 1
   Set nTax = 6
Else If nCountryID = 43
   Set nTax = 12
Else
   Set nTax = 10
Set nResult = nNet + ((nNet / 100 ) * nTax )
```

You can also use string expressions as an expression:

```
If sState = 'FL'
```

Note: The expression is case sensitive!

While

While acts as a loop that repeats until the expression being evaluated becomes FALSE.

The syntax is:

While Expression
    Statement

In the above example, TD Mobile evaluates Expression. If it is TRUE, Statement executes and TD Mobile re-evaluates Expression, and so on. When Expression becomes FALSE, TD Mobile resumes execution of the application at the action following the While statement.

Example:

```
If SqlPrepareAndExecute(hSQL, sSQL)
  While SqlFetchNext(hSQL, nRet)
    Set nIdx = nIdx + 1
```

The While block runs until SqlFetchNext returns FALSE.

Loop

Loop repeats any child statements indented under it until a Break or Return executes.

The syntax is:

    Loop [loop_name]

where the loop name is optional.

Example 1:

```
If SqlPrepareAndExecute(hSQL, sSQL)
  Loop
    Call SqlFetchNext(hSQL, nRet)
      If nRet = FETCH_EOF
        Break
```

Example 2:

```
If SqlPrepareAndExecute(hSQL, sSQL)
  Loop Outer
    Call SqlFetchNext(hSQL, nRet)
      If nRet = FETCH_EOF
        Break Outer
      Loop Inner
        Set nCounter = nCounter+ 1
          If nCounter = 100
            Break Inner
```

Select Case, Case, Default

Use Select Case when you have a series of conditions that you want to test.

With the Select Case statement, TD Mobile successively compares the value of an expression against Case constants. Both the expression and the constants must be number data types.

A Break statement signals the end of a Case, and terminates execution of the Select Case statement. You must have a Break at the end of each Case statement unless you want the program to continue execution through to the next Case.

The Default case is optional, and if it is present, it is placed at the end of the Select Case statement. It executes when the value of the expression does not match any of the case constants.

The syntax is:

```
Select Case (Expression)
        Case Constant1
                Statement1
                Break
        Case Constant2
                Statement2
                Break
        Default
                Statement3
```

In the above example, TD Mobile evaluates Expression. If its value matches that of Constant1, then Statement1 executes. If its value matches that of Constant2, then Statement2 executes. If no Case constant value matches that of Expression, then Statement3 executes.

You can specify as many Case constants as you want, but there can be only one Default section. Indentation determines the conditional flow of control. Use Break to terminate a Case.

To allow more than one Case constant to execute the same statement, stack them like this:

```
Select Case nZIP
  Case 33100
  Case 33000
    Set nSalesID = 1
    Break
  Case 40000
    Set nSalesID = 17
    Break
  Case 50000
    Set nSalesID = 22
    Break
  Case 70000
    Set nSalesID = 25
    Break
  Default
    Set nSalesID = 100
```

In the following example, TD Mobile evaluates SalDateQuarter (dtDate) and then sets strQuarter equal to the quarter of the year represented by this expression. If the expression does not evaluate to 1, 2, 3 or 4, strQuarter equals 'Unknown'.

```
Select Case ( SalDateQuarter ( dtDate ) )
        Case 1
                Set strQuarter = 'First Quarter'
                Break
        Case 2
```

```
        Set strQuarter = 'Second Quarter'
        Break
Case 3
        Set strQuarter = 'Third Quarter'
        Break
Case 4
        Set strQuarter = 'Fourth Quarter'
        Break
Default
        Set strQuarter = 'Unknown'
```

Connecting to a database

Starting with TD Mobile 1.1, basic data operations have been made very easy. It is possible to specify a Data Connection, define and map Data Classes to its data, and then Browse, Read, Edit, Add or Delete data with Data Operations – all without writing a single line of SAL code. You can read all about that in another TD Mobile whitepaper *NoSql DataConnections*.

However there may still be times when you need to write custom SQL, anytime you need to write joins, for example. Then you will need to use SAL.

SqlConnectDotNet

The SAL function for connecting to databases is SqlConnectDotNet. Here's an example when connecting to an Oracle database:

```
Local Variables
    Sql Handle: hSql
Actions
    If SqlConnectDotNet( hSql, "Data Source=MyOraServer;User Id=scott;Password=tiger",
                                "Oracle.DataAccess.Client", DBP_PROVIDER_ORACLE)
```

Let's look at the four arguments SqlConnectDotNet takes.

Arg#1: Sql Handle

The first argument the function takes, *hSql*, is an object of the SAL type **Sql Handle**. If the connection succeeds, *hSql* will represent a valid connection; you can then use it in subsequent calls to execute SQL, fetch records, commit transactions, disconnect, etc. We'll show some of these in a moment.

Arg#2: Connection String

A .NET connection string is a series of key=value pairs delimited by semicolons. The keys, like "Data Source" in the above example, are terms dictated by your data provider, while the values, like the name of your database server or your user name, are things known privately within your company. There are a lot of good examples in the documentation for *SqlConnectDotNet* in the SAL Help (in the TD Mobile Ribbon Bar, Help tab, the SAL icon is 2nd from the left). There are also good examples online, for example at www.connectionstrings.com

Arg#3: Invariant

The invariant string is a name that can be used programmatically to refer to the data provider. This string should correspond to the invariant attribute of the factory entry in the section of your machine.config. For example, if you had client software for MS SqlServer installed on your machine, you might have the following entry in your machine.config file:

```
<system.data>
  <DbProviderFactories>
    <add name="SqlClient Data Provider"
     invariant="System.Data.SqlClient"
     description=".Net Framework Data Provider for SqlServer"
     type="System.Data.SqlClient.SqlClientFactory, System.Data,
     Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
    />
  </DbProviderFactories>
</system.data>
```

The invariant name for SqlServer is “System.Data.SqlClient”.

Arg#4: Provider Type

Use one of these DBP_PROVIDER constants to identify your database provider:

- DBP_PROVIDER_UNDEFINED=0
- DBP_PROVIDER_SQLBASE_OLEDB=1
- DBP_PROVIDER_SQLSERVER_OLEDB=2
- DBP_PROVIDER_ORACLE_OLEDB=3
- DBP_PROVIDER_ODBC=4
- DBP_PROVIDER_ORACLE=5
- DBP_PROVIDER_OLEDB=7
- DBP_PROVIDER_SQLBASE=9
- DBP_PROVIDER_SQLSERVER_SQLCLIENT=12

SQL in a TD Mobile Operation

Let’s look at a fully expanded Operation that demonstrates connecting to a SqlBase database, executing a Select statement with a join, fetching all the records and then disconnecting.

All SQL-based SAL functions begin with the prefix ‘Sql’. We connect with *SqlConnectDotNet*; then we call *SqlPrepareAndExecute* to parse and execute the SQL; we then do a While loop of *SqlFetchNext* calls to select each row returned by the database; and finally we call *SqlDisconnect*. All of the Sql* functions return True if they succeed and False if they fail.

Practically speaking, the Operation is getting a list of Employees (whose salary is greater than the provided value *pMinSal*), selecting just the employee’s Id, name and department, and populating an array with that information; the operation returns that array back to the client, presumably, to show in a ListView control.

```
Operations
  Operation: GetEmployees
    Description:
    Parameters
      Number: pMinSal
        Binding: MIN_SAL
    Returns
      cEmployee: [*]
        Binding: EMPS
    Local Variables
      cEmployee: emps[*]
      String: firstName
      String: lastName
      Number: idx
      Sql Handle: hSql
      String: sSql
      Number: nFetch
    Actions
      If SqlConnectDotNet( hSql,
                "servername=server1;hostname=localhost;port=2155;database=island;
                        user=sysadm;password=sysadm;poolsize=10;connectionlifetime=20",
                "Gupta.SqlBase.Data", DBP_PROVIDER_SQLBASE)
        Set sSql = "SELECT E.employee_id, E.last_name, E.first_name, D.dept_name
                            FROM employee E, department D
                            WHERE E.dept_id = D.dept_id AND E.current_salary > :pMinSal
                            INTO :emps[idx].ID, :firstName, :lastName, :emps[idx].Department"
        If SqlPrepareAndExecute( hSql, sSql )
          While SqlFetchNext( hSql, nFetch )
            Set emps[idx].Name = firstName || " " || lastName
            Set idx = (idx + 1)
        Call SqlDisconnect( hSql )
      Return emps
```

SQL with Binds and Intos

Binds and Intos are variables, prefixed with a colon (":"), that are used within your SQL. TD Mobile parses them out and handles the variables' values for you.

The Bind variable, in the SQL in our example, is :pMinSal. It's actually a Parameter of the Operation, so scope-wise it's like a local variable. Bind variables are used as criteria in Where clauses, as in our example, or as values in Insert statements, for another example. They are data we want to pass to the database.

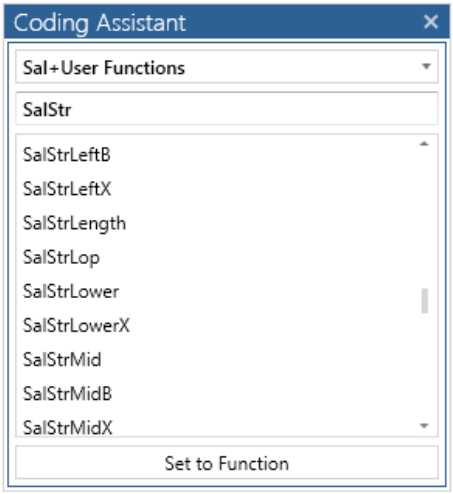
The Into variables, in our example, are identified directly by the 'INTO' statement; they are: :emps[idx].ID, :firstName, :lastName, :ems[idx].Department. Into variables are used to receive data from the database. With each call to SqlFetchNext, the Into variables are updated with the values from the current row of the returned recordset.

TD Mobile API

This section will describe some important Sal API functions in TD Mobile. This list is not complete. See Active Coding Assistant for more information. Also have a look into the Online Help of TD Mobile.

Following function groups are described at the following pages:

- Array functions
- File functions
- Number functions
- Date functions
- Debugging functions
- SQL functions
- SQL OLE DB functions
- String functions
- Object functions
- Miscellaneous functions



Array Functions

This is an alphabetical list of the SAL array functions accompanied by detailed information about each function's purpose, its parameters and return value, and an example.

Function descriptions include:

- Syntax
- Description
- Parameters
- Return value
- See also
- Example

SalArrayAvg

Syntax	***nAvg =* SalArrayAvg *(*** *nArrayNum)*
Description	Returns the average value of all the numbers in an array.
Parameters	nArrayNum Numeric Array. The name of an array of numbers.
Return Value	nAvg is the average value in an array of numbers.
See Also	SalArrayDimCount SalArrayGetLowerBound SalArrayGetUpperBound SalArrayIsEmpty SalArrayMax SalArrayMin SalArraySetUpperBound SalArraySum
Example	`Actions` `  Set dfAvg = SalArrayAvg( nArrayNum )`

SalArrayDimCount

Syntax ***bOk =* SalArrayDimCount** *(aArray, nDim)*

Description Returns the number of dimensions in an array.

Parameters

aArray	Array. The name of the array to query.
nDim	Receive Number. Number of dimensions in the array.

Return Value bOk is TRUE if the function succeeds and FALSE if it fails.

See Also SalArrayAvg SalArrayGetLowerBound SalArrayGetUpperBound SalArrayIsEmpty SalArrayMax SalArrayMin SalArraySetUpperBound SalArraySum

Example

```
Actions
  Set bOk = SalArrayDimCount (aArray, nDim) If nDim = 0
  Set bDimTrue = False
```

SalArrayGetLowerBound

Syntax ***bOk =* SalArrayGetLowerBound** *(aArray, nDim, nBound)*

Description Returns the lower bound of an array.

Parameters

aArray	Array. The name of the array to query.
nDim	Number. Number of the dimension to query. The first dimension is one, the second is two, and so on.
nBound	Receive Number. Lower bound value.

Return Value bOk is TRUE if the function succeeds and FALSE if it fails.

See Also SalArrayAvg SalArrayDimCount SalArrayGetUpperBound SalArrayIsEmpty SalArrayMax SalArrayMin SalArraySetUpperBound SalArraySum

Example

```
Actions
  Set bOk = SalArrayGetLowerBound (aArray, nBound)
```

SalArrayGetUpperBound

Syntax ***bOk =* SalArrayGetUpperBound** *(aArray, nDim, nBound)*

Description Returns the upper bound of an array.

Parameters

aArray	Array. The name of the array to query.
nDim	Number. Number of the dimension to query. The first dimension is one, the second is two, and so on.
nBound	Receive Number. Upper bound value.

Return Value bOk is TRUE if the function succeeds and FALSE if it fails.

See Also SalArrayAvg SalArrayDimCount SalArrayGetLowerBound SalArrayIsEmpty SalArrayMax SalArrayMin SalArraySetUpperBound SalArraySum

Example

```
Actions
```

```
Set bOk = SalArrayGetUpperBound (aArray, nBound)
```

SalArrayIsEmpty

Syntax ***bData* = SalArrayIsEmpty** *(aArray)*

Description Determines if a dynamic array contains data.

Parameters aArray Array. The name of the array to query.

Return Value bData is TRUE if the array contains no data and FALSE if it is has data.

See Also SalArrayAvg SalArrayDimCount
SalArrayGetLowerBound SalArrayGetUpperBound
SalArrayMax SalArrayMin SalArraySetUpperBound
SalArraySum

Example
```
Actions
  If Not SalArrayIsEmpty (aArray) Call ...
```

SalArrayMax

Syntax ***nMax* = SalArrayMax** *(nArrayNum)*

Description Returns the maximum value in an array of numbers.

Parameters nArrayNum Numeric Array. The name of an array of numbers.

Return Value nMax is the maximum value in an array of numbers.

See Also SalArrayAvg SalArrayDimCount
SalArrayGetLowerBound SalArrayGetUpperBound
SalArrayIsEmpty SalArrayMin
SalArraySetUpperBound SalArraySum

SalArrayMin

Syntax ***nMin* = SalArrayMin** *(nArrayNum)*

Description Returns the minimum value in an array of numbers.

nArrayNum Numeric Array. The name of an array of numbers.

Return Value nMin is the minimum value in an array of numbers.

See Also SalArrayAvg SalArrayDimCount
SalArrayGetLowerBound SalArrayGetUpperBound
SalArrayIsEmpty SalArrayMax
SalArraySetUpperBound SalArraySum

Example
```
Actions
    Set dfMin = SalArrayMin( nArrayNum )
```

SalArraySetUpperBound

Syntax ***bOk* = SalArraySetUpperBound** *(aArray, nDim, nBound)*

Description Sets the upper bound of an array.

When you call this function for a dimension other than the first, TD Mobile must copy most of the array's data. There can be a performance cost when you call this function for any dimension but the first one.

Parameters

aArray	Array. The name of the array to query.
nDim	Number. Number of the dimension to query. The first dimension is one, the second is two, and so on.
nBound	Number. Upper bound value. Specify -1 to reset a dynamic array to zero elements. Specify AC_Dynamic to change a static array to a dynamic array. (You can make only the first dimension of an array dynamic.)

Return Value

bOk is TRUE if the function succeeds and FALSE if it fails.

See Also

SalArrayAvg SalArrayDimCount SalArrayGetLowerBound SalArrayGetUpperBound SalArrayIsEmpty SalArrayMax SalArrayMin SalAraySum

Example

```
Actions
      Call SalArraySetUpperBound (aArray, 1, 5)
```

SalArraySum

Syntax

***nSum* = SalArraySum** *(nArrayNum)*

Description

Returns the sum of the elements in an array of numbers.

Parameters

nArrayNum	Numeric Array. The name of an array of numbers.

Return Value

nSum is the sum of the elements in an array of numbers.

See Also

SalArrayAvg SalArrayDimCount SalArrayGetLowerBound SalArrayGetUpperBound SalArrayIsEmpty SalArrayMax SalArrayMin SalArraySetUpperBound

Example

```
Actions
      Set dfSum = SalArraySum( nArrayNum )
```

Date Functions

This is an alphabetical list of the SAL date functions accompanied by detailed information about each function's purpose, its parameters and return value, and an example.

Function descriptions include:

- Syntax
- Description
- Parameters
- Return value
- See also

- Example

SalDateConstruct

Syntax ***dtDate* = SalDateConstruct** *(nYear, nMonth, nDay, nHour, nMinute, nSecond)*

Description Returns the date/time constructed from the parameters nYear, nMonth, nDay, nHour, nMinute, and nSecond.

If you specify invalid parameter values, an unexpected date construction can result.

Note: If any of the parameter values has less than the specified number of digits, SalDateConstruct pads the resulting value with leading zeroes (0). For example, if nYear is 92, dtDate begins with 0092.

Parameters

nYear	Number. A number with a 4-digit year value greater than zero.
nMonth	Number. A number with a 2-digit month value between 01 and 12 inclusive.
nDay	Number. A number with a 2-digit day value between 01 and 31 inclusive.
nHour	Number. A number with a 2-digit hour value between 0 and 23 inclusive.
nMinute	Number. A number with a 2-digit minute value between 01 and 59 inclusive.
nSecond	Number. A number with a 2-digit second value between 01 and 59 inclusive.

Return Value dtDate is the newly constructed date/time value.

Example

```
Actions
    Set dtDeb = SalDateConstruct (1996, 1, 1, 10, 30, 1)
```

SalDateCurrent

Syntax ***dtNow* = SalDateCurrent** *( )*

Description Returns the PC's current date/time.

To get the database server's system date/ time, use a database system keyword (SYSDATE, SYSTIME, or SYSDATETIME) in a SQL SELECT statement.

Parameters No parameters.

Return Value dtNow is the PC's current date/time.

Example

```
Actions
      Set dfCurrent = SalDateCurrent ( )
```

SalDateDay

Syntax ***nDay* = SalDateDay** *(dtDateTime)*

Description Returns the day portion (1 to 31) of a date/time value or returns -1 if you specify DATETIME_Null as a parameter.

Parameters dtDateTime Date/Time. A date/time value.

Return Value nDay is a number between 1 and 31.

See Also SalDateMonth

SalDateYear

Example
```
Set nDay = SalDateDay ( dtDateTime )
```

SalDateHour

Syntax ***nHour =*** **SalDateHour** *(dtDateTime)*

Description Returns the hour portion (0 to 23) of a date/time value or returns -1 if you specify DATETIME_Null as a parameter.

Parameters dtDateTime Date/Time. A date/time value.

Return Value nHour is a number between 0 and 23.

See Also SalDateMinute

SalDateSecond

Example
```
Set nHour= SalDateHour ( dtDateTime )
```

SalDateMinute

Syntax ***nMinute =*** **SalDateMinute** *(dtDateTime)*

Description Returns the minute portion (0 to 59) of a date/time value or returns -1 if you specify DATETIME_Null as a parameter.

Parameters dtDateTime Date/Time. A date/time value.

Return Value nMinute is a number between 0 and 59.

See Also SalDateHour

SalDateSecond

Example
```
Set nMinute = SalDateMinute ( dtDateTime )
```

SalDateMonth

Syntax ***nMonth =*** **SalDateMonth** *(dtDateTime)*

Description Returns the month portion (1 to 12) of a date/time value or returns -1 if you specify DATETIME_Null as a parameter.

Parameters dtDateTime Date/Time. A date/time value.

Return Value nMonth is a number between 1 and 12.

Related Functions

SalDateDay
SalDateYear

Example
```
Set nMonth = SalDateMonth ( dtDateTime )
```

SalDateMonthBegin

Syntax *dtMonthBegin* = **SalDateMonthBegin** *(dtDateTime)*

Description Returns the date of the first day of the month or it returns DATETIME_Null if the value you specify is null. For example, if dtDateTime is December 25, 1992, SalDateMonthBegin returns December 1, 1992.

Parameters dtDateTime Date/Time. A date/time value.

Return Value dtMonthBegin is the date of the first day of the month of dtDateTime.

See Also SalDateQuarterBegin
SalDateWeekBegin

Example
```
Set dtMonthBegin = SalDateMonthBegin ( SalDateCurrent ( ) )
```

SalDateQuarter

Syntax ***nQuarter* = SalDateQuarter** *(dtDateTime)*

Description Returns the quarter of the year (1 to 4) of a date/time value or returns -1 if you specify DATETIME_Null as a parameter.

Parameters dtDateTime Date/Time. A date/time value.

Return Value nQuarter is a number between 1 and 4.

Example
```
Select Case ( SalDateQuarter ( dtDate ) ) Case 1
       Set strQuarter = 'First Quarter' Break
       ...
```

SalDateQuarterBegin

Syntax ***dtQuarterBegin* = SalDateQuarterBegin** *(dtDateTime)*

Description Returns the date of the first day of the quarter of a date/time value or it returns DATETIME_Null if the value you specify is null.

Parameters dtDateTime Date/Time. A date/time value.

Return Value dtQuarterBegin is the first day of the quarter of dtDateTime.

See Also SalDateMonthBegin SalDateWeekBegin
SalDateYearBegin

Example
```
Set dtQuarterBegin = SalDateQuarterBegin
      ( SalDateCurrent ( ) )
```

SalDateSecond

Syntax ***nSeconds* = SalDateSecond** *(dtDateTime)*

Description Returns the seconds portion (0 to 59) of a date/time value or returns -1 if you specify DATETIME_Null as a parameter.

Parameters dtDateTime Date/Time. A date/time value.

Return Value nSeconds is a number between 0 and 59.

See Also SalDateHour
SalDateMinute

Example
```
Set nSeconds = SalDateSecond ( dtDateTime )
```

SalDateToStr

Syntax ***nLength =*** **SalDateToStr** *(dtDateTime, strDate)*

Description Converts a date/time value to a string value or returns -1 if you specify DATETIME_Null as a parameter.

Parameters dtDateTime Date/Time. The date/time value to convert. strDate Receive string. The resulting string value.

Return Value nLength is the length of strDate.

See Also SalFmtFormatDateTime
SalStrToDate

Example
```
Call SalDateToStr ( dtDateTime, strDateTime )
    ! strDateTime = YYYY-MM-DD-HH.MM.SS.TTTTTT
```

SalDateWeekBegin

Syntax ***dtWeekBegin* = SalDateWeekBegin** *(dtDateTime)*

Description Returns the date of the previous Monday or the current day if it is a Monday or it returns DATETIME_Null if the value you specify is null.

Parameters dtDateTime Date/Time. A date/time value.

Return Value dtWeekBegin is the date of the previous Monday, or today's date if it is Monday.

See Also SalDateMonthBegin SalDateQuarterBegin SalDateYearBegin

Example
```
Set dtWeekBegin = SalDateWeekBegin ( SalDateCurrent ( ) )
```

SalDateWeekday

Syntax ***nWeekday* = SalDateWeekday** *(dtDateTime)*

Description Returns the day of the week as a number between 0 and 6 or returns -1 if you specify DATETIME_Null as a parameter. 0 represents Saturday, 1 represents Sunday, and so on.

Parameters dtDateTime Date/Time. A date/time value.

Return Value nWeekday is a number between 0 and 6.

Example
```
Select Case ( SalDateWeekday ( dtDate ) ) Case 0
       Set strWeekday = 'Saturday'
       Break
```

SalDateYear

Syntax ***nYear* = SalDateYear** *(dtDateTime)*

Description Returns the year portion of a date or returns -1 if you specify DATETIME_Null as a parameter.

Parameters dtDateTime Date/Time. A date/time value.

Return Value nYear is the year portion of a date.

See Also SalDateDay SalDateMonth

Example
```
Set nYear = SalDateYear ( dtDateTime )
```

SalGetDateTime

Syntax ***dtDateTime* = SalGetDateTime** *(hWnd)*

Description This api returns the date/date time value stored in the date picker or the date time picker.

Parameters

hWnd	Window Handle. The handle (or name) of the 'Date Picker' or 'Date Time Picker' control.

Return Value dtDateTime - This is the date/datetime value stored in the 'Date Picker' or 'Date Time Picker'

See Also SalSetDateTime

```
Set nDataType = SalGetDataType ( hWndChild ) Set hWndSave  =
SalGetFocus ( )
```

Debugging Functions

This is an alphabetical list of the SAL debugging functions accompanied by detailed information about each function's purpose, its parameters and return value, and an example.

Function descriptions include:

- Syntax
- Description
- Parameters
- Return value
- See also
- Example

SalCompileAndEvaluate

Syntax ***nType* = SalCompileAndEvaluate** *(strExpression, nError, nErrorPos, nReturn, strReturn, dtReturn, hWndReturn, bInhibitErrors, strContext)*

Description Evaluates an expression and returns the expression's value in the receive parameter appropriate to its data type. SalCompileAndEvaluate lets you access the value of a variable whose name you do not specify until runtime.

Parameters

strExpression	String. The expression to evaluate.
nError	Receive Number. The error number, if one is returned.
nErrorPos	Receive Number. The position in strExpression at which an error, if any, occurred.
nReturn	Receive Number. This parameter is set if strExpression evaluates to a number.
strReturn	Receive String. This parameter is set if strExpression evaluates to a string.
dtReturn	Receive Date/Time. This parameter is set if strExpression evaluates to a date/time value.
hWndReturn	Receive Window Handle. This parameter is set if strExpression evaluates to

a handle.

bInhibitErrors	Boolean. If TRUE, TD Mobile does not report compilation or evaluation errors to the user. Specify TRUE if the application processes its own errors. If FALSE, TD Mobile reports compilation and evaluation errors to the user in a dialog box.
strContext	String. The handle to an execution context, returned by either SalContextBreak or SalContextCurrent.

Return Value nType is equal to one of the following values if the function succeeds: EVAL_Date EVAL_Handle EVAL_If EVAL_Number EVAL_Set EVAL_String EVAL_Template

See Also SalContextBreak
SalContextCurrent

Example
```
Set nType = SalCompileAndEvaluate ( strExpression, nError, nErrorPos,nReturn,
     strReturn, dtReturn, hWndReturn, FALSE, strContext )
If nType = EVAL_Number
     Call SalNumberToStr ( nReturn, 0, strString )...
```

SalContextBreak

Syntax ***strContext* = SalContextBreak** *( )*

Description Retrieves the context of the most recently executed Break statement. Use this function with SalCompileAndEvaluate.

Parameters No parameters.

Return Value strContext serves as the last parameter of the SalCompileAndEvaluate function.

See Also SalCompileAndEvaluate
SalContextCurrent

Example
```
Set strContext = SalContextBreak ( )
```

SalContextCurrent

Syntax ***strContext* = SalContextCurrent** *( )*

Description Retrieves the current execution context. Use this function with SalCompileAndEvaluate.

Parameters No parameters.

Return Value strContext serves as the last parameter of the SalCompileAndEvaluate function.

See Also SalCompileAndEvaluate
SalContextBreak

Example
```
Set strContext = SalContextCurrent ( )
```

SalEndTrace

Syntax **SalEndTrace** *( )*

Description Ends all tracing. Any calls to SalTrace() made after this function is called will be ignored.

Parameters	none
Return Value	none.
See Also	SalStartTrace SalTrace
Example	No example

SalStartTrace

Syntax ***bOk* = SalStartTrace *(nOutputType, strTraceFile, bClearExisting)***

Description Allows tracing to begin.

You are responsible for ensuring that existing trace log files do not grow too large.

Parameters

nOutputType	Number. One of the four TRACE_* constants: *TRACE_Event* outputs information to the Windows event log. (Windows 98 and Windows ME do not have event logging as a built-in operating system feature. In these cases, when TRACE_Event is chosen for nOutputType, the trace information goes to file "TDEvent.log" in the Windows temporary directory.) *TRACE_File* outputs to the file named in the strTraceFile parameter. *TRACE_Output* outputs to the TD Mobile output window, which ordinarily displays information such as compile-time errors. This option only works when the TD Mobile application is in debug mode. *TRACE_stdout* outputs to the standard output device; it is designed to make trace output available to third-party diagnostic applications
strTraceFile	String. The name of the file to receive output when nOutputType is TRACE_File. If that output type is chosen but this parameter is left null, a file will be created in the Windows temporary directory. The file name will be the name of the application executable, with a suffix of .LOG.
bClearExisting	Boolean. Whether existing output should be cleared before new tracing begins..

Return Value bOk is TRUE if the functions succeeds, and FALSE if it fails.

See Also SalTrace
SalEndTrace

Example

```
Call SalStartTrace ( TRACE_Event, '', TRUE )

all SalStatusSetVisible ( hWndForm, TRUE )
```

SalTrace

Syntax ***bOk* = SalTrace *(nSeverity, strTextToWrite)***

Description Writes a string of text to the trace output target that was specified in an earlier call to SalStartTrace.

Parameters

nSeverity	Number. One of the following three constants: EVENT_Error, EVENT_Warning, or EVENT_Information. When used with the Windows event log, these numeric values will be integrated into that log's severity system. When output is going to some other target, such as a file, these numeric values are translated into text strings.

strTextToWrite String. The text of the trace message.

Return Value bOk is TRUE if the function succeeds and FALSE if it fails.

See Also SalStartTrace

SalEndTrace

Example

```
bOk = SalTrace (EVENT_Warning, 'User entered a null password')
```

File Functions

This is an alphabetical list of the SAL file functions accompanied by detailed information about each function's purpose, its parameters and return value, and an example.

Function descriptions include:

- Syntax
- Description
- Parameters
- Return value
- See also
- Example

SalFileClose

Syntax ***bOk* = SalFileClose** *(hFile)*

Description Closes a file.

Parameters hFile Receive file handle. The handle of the file to close. When the function returns, the value of this parameter becomes null.

Return Value bOk is TRUE if the function succeeds and FALSE if it fails.

See Also SalFileOpen

Example

```
If NOT SalFilePutStr ( hFile, sLine ) Call SalFileClose (
     hFile )
     Call SalMessageBeep ( 0 )
```

SalFileCopy

Syntax ***nStatus* = SalFileCopy** *(strSourcePath, strDestPath, bOverWrite)* Description Copies the contents of one file (source) to another file (destination). Parameters strSourcePath String. The full path name of the source file.

strDestPath String. The full path name of the destination file.

	bOverWrite	Boolean. Specifies whether (TRUE) or not (FALSE) to overwrite the destination file. If the destination file already exists and bOverWrite is FALSE, then SalFileCopy fails, and returns FILE_CopyExist. If the destination file already exists and bOverWrite is TRUE, then SalFileCopy succeeds and the destination file is overwritten.

Return Value nStatus is equal to one of the following values: FILE_CopyDest
FILE_CopyExist FILE_CopyOK
FILE_CopyRead FILE_CopySrc
FILE_CopyWrite

Example

```
Actions
     Set bLogFileSaved = SalFileCopy ( 'C:\\DB\\APP.LOG', ('C:\\DB\\APP.OLD',
        TRUE )
```

SalFileCreateDirectory

Syntax ***bOk =*** **SalFileCreateDirectory** *(strDir)*

Description Creates a directory.

Parameters

	strDir	String. The full path name of the new directory.

Return Value bOk is TRUE if the function succeeds and FALSE if a directory or file with the specified name already exists, or if the specified path to the directory cannot be found.

See Also SalFileRemoveDirectory

Example

```
Set DirCreated = SalFileCreateDirectory('C:\\NOTES\\REL2' )
```

SalFileGetC

Syntax ***bOk =*** **SalFileGetC** *(hFile, nChar)*

Description Returns the next character in an open file. You must use this function in place of the SalFileGetChar function if the file contains non-ASCII (ANSI) or 16-bit characters.

If the character returned is a 16-bit character, the lead byte of the character is in the high-order byte, and the trail byte is in the low-order byte. To get the lead byte, use SalNumberHighand to get the trail byte, use SalNumberLow.

Parameters hFile File Handle. The handle of the open file. nChar Receive Number. The next character in hFile.

Return Value bOk is TRUE if the function succeeds and FALSE if the function is unable to read the next character from the file, or if an invalid file handle is passed in hFile.

See Also SalFilePutC

Example

```
Actions
   Call SalFileOpen ( hFile, 'C:\\DB\\APP.LOG', OF_Read | OF_Binary
     )
```

SalFileGetChar

Syntax ***nChar =*** **SalFileGetChar** *(hFile)*

Description Returns the next character in an open file.

Parameters	hFile	File Handle. The handle of the open file.
Return Value	nChar is a number that represents an ANSI character. At the end of the file, SalFileGetChar returns a -1.	
See Also	SalFilePutChar	

Example

```
If SalFileOpen ( fhDestFile, strDestFile, OF_Create
        | OF_ReadWrite ) Loop
        Set nChar = SalFileGetChar ( fhSrcFile )
    ...
```

SalFileGetCurrentDirectory

Syntax ***bOk* = SalFileGetCurrentDirectory** *(strPath)*

Description	Gets the full path name of the current working directory.	
Parameters	strPath	Receive String. The full path name, including the drive letter, of the current working directory.
Return Value	bOk is TRUE if the function succeeds and FALSE if it fails.	
See Also	SalFileSetCurrentDirectory	

Example

```
Actions
   If NOT SalFileGetCurrentDirectory  (strCurrentDir) Call SalMessageBox
      ( 'Couldn't get current
                   directory', 'Error', 0)
```

SalFileGetDateTime

Syntax ***bOk* = SalFileGetDateTime** *(strFilename, dtDateTime)*

Description	Gets the modification date and time of the specified file.	
Parameters	strFilename	String. The name of the file whose modification date you want.
	dtDateTime	Receive Date/Time. The modification date and time of strFilename.
Return Value	bOk is TRUE if the function succeeds and FALSE if it fails.	
See Also	SalFileSetDateTime	

Example

```
Set bOk = SalFileGetDateTime ( 'WIN.INI', gdFileDate )
```

SalFileGetDrive

Syntax ***strDriveLetter* = SalFileGetDrive** *( )* Description Gets the letter of the default (current) disk drive. No Parameters

Return Value	strDriveLetter is a string identifying the current disk drive. The first character is a letter between 'A' and 'Z', and the second character is a colon(:).
See Also	SalFileSetDrive

Example

```
Actions
```

```
If SalFileGetDrive ( ) = 'A' Set blsDriveA =
      TRUE
Else
  Set blsDriveA = FALSE
```

SalFileGetStr

Syntax ***bOk =* SalFileGetStr** *(hFile, strBuffer, nBufferSize)*

Description Returns the next line from an open file. TD Mobile strips off the trailing carriage return/line feed of the returned string.

Parameters hFile File Handle. The handle of the open file. strBuffer

Receive String. The returned string.

nBufferSize Number. The maximum number of bytes to read.

Return Value bOk is TRUE if the file is successfully read and FALSE otherwise. FALSE is also returned on end of file.

See Also SalFilePutStr

Example
```
If SalFileGetStr ( fhSrcFile, strText, LINE_SIZE ) Call SalFilePutStr
     ( fhDestFile, strText )
Else
     Break
```

SalFileOpen

Syntax ***bOk =* SalFileOpen** *(hFile, strFileName, nStyle)*

Description Opens, re-opens, creates, or deletes a file.

Parameters hFile Receive File Handle. The handle of the open file. strFileName String. The name of the file to open, create, delete, or test.

nStyle Number. A constant that specifies the style in which to open the file. nStyle can be one or more styles combined using the bitwise OR (|) operator.

Return Value bOk is TRUE if the function succeeds and FALSE if it fails.

See Also SalFileClose `SalListFiles`
`SalFileOpenExt`

Example
```
Call SalFileOpen ( fhSrcFile, strSrcFile, OF_Read )
```

SalFileOpenExt

Syntax ***bOk =* SalFileOpenExt** *(hFile, strFileName, nStyle, strReopen)*

Description Opens or re-opens a file. Long filenames up to 260 characters is supported.

Parameters hFile Receive File Handle. The handle of the opened or re-opened file.

strFileName String. The name of the file to open, create, delete, or test.

	nStyle	Number. A constant that specifies the style in which to open the file. nStyle can be one or more styles combined using the OR (|) operator.
	strReopen	Receive String. Information used to re-open the file.

Return Value bOk is TRUE if the function succeeds and FALSE if it fails.

See Also SalFileOpenF

Example
```
Actions
    If SalFileOpenExt ( hFile, 'C:\\AUTOEXEC.BAT', OF_Read, strReopen )
```

SalFilePutC

Syntax ***bOk =* SalFilePutC** *(hFile, nChar)*

Description Writes a character to an open file. Use this function instead of SalFilePutChar if the character is a non-ASCII (ANSI) or 16-bit character.

Parameters

	hFile	File Handle. The handle of the open file.
	nChar	Number. The non-ASCII or 16-bit numeric value of the character to write to hFile.

Return Value bOk is TRUE if the function succeeds and FALSE if it is unable to write to hFile.

See Also SalFileGetC

Example
```
Loop
    Call SalFilePutC ( hFile, nNull )
    If nCount = 5
                        Break
              Set nCount = nCount + 1
```

SalFilePutChar

Syntax ***bOk =* SalFilePutChar** *(hFile, nChar)*

Description Writes a character to an open file.

Parameters

	hFile	File Handle. The handle of the open file.
	nChar	Number. The ANSI numeric value of the character to write to hFile.

Return Value bOk is TRUE if the function succeeds and FALSE if it fails.

See Also SalFileGetChar

Example
```
Call SalFileSeek (fhInFile, nFilePos, FILE,_SeekBegin) Call SalFilePutChar
( hFile, nChar )
```

SalFilePutStr

Syntax ***bOk =* SalFilePutStr** *(hFile, strString)*

Description Writes a string to an open file. TD Mobile appends a carriage return/line feed character to the string.

Parameters

	hFile	File Handle. The handle of the open file. strString String.

The string to write.

Return Value bOk is TRUE if the function succeeds and FALSE if it fails.

See Also SalFileGetStr

SalFileRead

Syntax ***nResult =* SalFileRead** *(hFile, strBuffer, nBufferLength)*

Description Reads a buffer of characters from an open file to a string .

Parameters hFile File Handle. The handle of the open file.

strBuffer Receive String. The string to which the data is read. nBufferLength

Number. The number of bytes to read.

Return Value nResult is the number of bytes read. On end of file, SalFileRead returns a byte count less than the requested amount.

See Also SalFileWrite

Example
```
Call SalFileSeek ( fhInFile, nFilePos, FILE_SeekBegin ) Loop
    Set nCharsRead = SalFileRead ( fhInFile, strBuffer, nRecSize )
```

SalFileRemoveDirectory

Syntax bOk = SalFileRemoveDirectory (strDir)

Description Deletes a directory.

Parameters strDir String. The full path name of the directory to delete.

Return Value bOk is TRUE if the function succeeds and FALSE if it fails. SalFileRemoveDirectory also returns FALSE if strDir contains files or other directories.

See Also SalFileCreateDirectory

Example Actions
Set bNotesOldDeleted = SalFileRemoveDirectory (strDir)

SalFileSeek

Syntax ***bOk =* SalFileSeek** *(hFile, nBytes, nPosition)*

Description Positions the file pointer in an open file. The next file operation (such as a read or write) takes place at this new location.

Parameters

hFile	File Handle. The handle of an open file.
nBytes	Number. The specific position of the file pointer; the number of bytes from nPosition where the next file operation will take place.
nPosition	Number. The general position of the file pointer; one of the following values:

```
FILE_SeekBegin
FILE_SeekCurrent
FILE_SeekEnd
```

Return Value bOk is TRUE if the function succeeds and FALSE if it fails.

See Also SalFileTell

Example
```
Call SalFileSeek ( fhInFile, 0, FILE_SeekBegin )
```

SalFileSetCurrentDirectory

Syntax ***bOk = SalFileSetCurrentDirectory*** *(strPath)*

Description Changes the current working directory. If the specified path does not contain a drive letter, the default drive's current directory is changed. Otherwise, the specified drive's current directory is changed without making that drive current.

Parameters strPath String. The path name of the new current working directory.

Return Value bOk is TRUE if the function succeeds and FALSE if it fails.

See Also SalFileGetCurrentDirectory
SalFileSetDrive

Example
```
Actions
    Set DirOk = SalFileSetCurrentDirectory ('C:\\NOTES\\REL2' )
```

SalFileSetDateTime

Syntax ***bOk =*** **SalFileSetDateTime** *(strFilename, dtDateTime)*

Description Sets the modification date and time of the specified file.

Parameters strFilename String. The name of the file whose modification date you want to set.

dtDateTime Date/Time. The modification date and time.

Return Value bOk is TRUE if the function succeeds and FALSE if it fails.

See Also SalFileGetDateTime

Example
```
Actions
   Set bOk = SalFileSetDateTime ('SQL.INI', SalDateCurrent ( ))
```

SalFileSetDrive

Syntax ***bOk =*** **SalFileSetDrive** *(strDriveLetter)*

Description Sets the current disk drive to the specified drive letter.

Parameters strDriveLetter String.0 The new disk drive letter. The length of this parameter's value is one character. If you specify a value larger than this, TD Mobile reads only the first character.

Return Value bOk is TRUE if the function succeeds and FALSE if it fails.

See Also SalFileGetDrive
SalFileSetCurrentDirectory

Example
```
Actions
     Set bDriveIsC = SalFileSetDrive ( 'c' )
```

SalFileTell

Syntax ***nPos* = SalFileTell** *(hFile)*

Description Returns the current position in an open file.

Parameters hFile File Handle. The handle of an open file.

Return Value nPos is the current position in hFile. If an error occurs, nPos is equal to -1.

See Also SalFileSeek

Example
```
Set nRecPos = SalFileTell ( fhInFile)
```

SalFileWrite

Syntax ***nResult* = SalFileWrite** *(hFile, strBuffer, nBufferLength)*

Description Writes a string to an open file.

Parameters hFile File Handle. The handle of an open file. strBuffer String.

The string to write to hFile. nBufferLength Number. The number of bytes to write.

Return Value nResult is the number of bytes written.

See Also SalFileRead

Example

```
Call SalFileWrite ( fhInFile, strCharBuf, 1 )
```

Miscellaneous Functions

This is an alphabetical list of the SAL functions accompanied by detailed information about each function's purpose, its parameters and return value, and an example.

Function descriptions include:

- Syntax
- Description
- Parameters
- Return value
- See also
- Example

SalGetBufferLength

Syntax ***nLength* = SalGetBufferLength** *(sTargetStr)*

Description Retrieves the number of bytes used by the buffer to store a specified string. Parameters sTargetStr

String. The string you want to get the storage buffer length for. Return Value nLength is the number of bytes.

See Also SalSetBufferLength

Example

```
Set nBuffLength = SalGetBufferLength ('Peter')
```

SalGetProfileInt

Syntax ***nValue* = SalGetProfileInt** *(strSection, strEntry, nDefault, strFileName)*

Description Retrieves the integer value of an entry in the specified section of an initialization file or registry.

Parameters

strSection	String. The section heading.
strEntry	String. The entry whose associated value is being retrieved.
nDefault	Number. Specify the default value (0 to 32,767) to return if the function cannot find the entry.
strFileName	String. The name of the initialization file or company name depending on the settings made using the SalUseRegistry function. If you are searching for an INI file and do not specify the full path, TD Mobile searches for the file in the Windows subdirectory.

Return Value nValue is the integer value of an entry in the specified section of a file or registry, if the function is successful. If the value found is not an integer, nValue is zero (0). If SalGetProfileInt cannot find the specified entry, nValue is the default value of the entry.

See Also SalGetProfileString SalSetProfileString SalUseRegistry

Example

```
Set nValue = SalGetProfileInt ( strSection, strEntry, nDefault, strFileName )
```

SalGetProfileString

Syntax ***nBytes* = SalGetProfileString** *(strSection, strEntry, strDefault, strValue, strFileName)*

Description Retrieves the string value of an entry in the specified section of an initialization file or registry.

Parameters

strSection	String. The section heading.
strEntry	String. The entry whose associated value is being retrieved.
strDefault	String. Specify the default value to return if the function cannot find the entry.
strValue	Receive String. The value of strEntry. Maximum 1024 bytes.
strFileName	String. The name of the initialization file or company name depending on the settings made using the SalUseRegistry function. If you are searching for an INI file and do not specify the full path, TD Mobile searches for the file in the Windows subdirectory.

Note: Specify the strFileName parameter as a NULL string when a company name is not necessary.

Return Value nBytes is the number of bytes copied to strValue, not including the terminating null character.

See Also SalGetProfileInt SalSetProfileString SalUseRegistry

Example

```
Set nBytes = SalGetProfileString ( strSection, strEntry, strDefault, strValue,
strFileName )
```

SalSetBufferLength

Syntax ***bOK* = SalSetBufferLength** *(sTargetStr,nBuffLength)*

Description Sets the number of bytes used by the buffer to store a specified string.

Parameters

sTargetStr	The string variable you want to set the storage buffer length for.
nBuffLength	The number of storage bytes used for the string variable

Return Value bOK is a boolean which returns true if the api is successful and false if not.

See Also SalGetBufferLength

Example

```
String sTest
Call SalSetBufferLength (sTest,5)
```

SalSetProfileString

Syntax ***bOk* = SalSetProfileString** *(strSection, strEntry, strValue, strFileName)*

Description Set the value of an entry in the specified section of an initialization file or registry. All profile information is stored as string, so if you want to store an integer it must be converted to a string first. Then it can be retrieved as an integer using the SalGetProfileInt-Function .

Parameters

strSection	String. The section heading.
strEntry	String. The entry whose associated value is being set. strValue String. The value of strEntry.
strFileName	String. The name of the initialization file or company name depending on the settings made using the SalUseRegistry function. If you are searching for an INI file and do not specify the full path, TD Mobile searches for the file in the Windows subdirectory.

Return Value bOk is TRUE if the function succeeds and FALSE if it fails.

See Also SalGetProfileInt SalGetProfileString
SalUseRegistry

Example

```
Set bOk = SalSetProfileString ( strSection, strEntry, strValue, strFileName )
```

SalUseEventLog

Syntax

bOk = **SalUseEventLog***(bUseEventLog, bContinueProcessing)*

Description

This function is used to start redirecting TD Mobile event processing from pop-up message boxes to the Windows event log, or to end such redirection. Some operating systems do not support true Windows event logging. See *Event Logging* in Chapter 10 of *Developing with TD Mobile* for more information.

When this function is called with bUseEventLog=TRUE, TD Mobile checks for a registry key and creates it if it does not already exist. This key is: HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\EventLog\Application\SWMSG40

Note: If the user running the application does not have Administrator privileges, the creation of the registry key will fail.

Parameters

bUseEventLog — TRUE if event log is to be used, FALSE if pop-up message boxes are to be used.

bContinueProcessing — Indicates whether to attempt to automatically resume the application when a recoverable SQL event occurs. Such an event would be one that would contain a Yes or Continue pushbutton if displayed in a pop-up message dialog box. To get this behavior, set this parameter to TRUE. To cause execution to stop for recoverable SQL events, set this parameter to FALSE. If a *non-SQL event* occurs, execution always stops, regardless of the setting of bContinueProcessing.

Return Value

bOk is TRUE if the function succeeds and FALSE if function fails.

Example

```
Set bOk = SalUseEventLog( TRUE, TRUE )
```

SalUseRegistry

Syntax

bOk = **SalUseRegistry***(bUseRegistry, sCompanyName)*

Description

This function is used to re-direct all SalProfile* functions to use the Registry. All values for the SalProfile* functions are saved as strings. The INI file name used as the last parameter of the SalProfile* functions will be used as the application name. The resulting path will be:

```
\\HKEY_CURRENT_USER\Software\<company name>\<application
name>\<section>\<setting>\
```

Parameters

bUseRegistry - TRUE if registry is to be used, FALSE if INI files are to be used.

sCompanyName - Name of company to be used under the registry.

Number Functions

This is an alphabetical list of the SAL number functions accompanied by detailed information about each function's purpose, its parameters and return value, and an example.

Function descriptions include:

- Syntax
- Description
- Parameters
- Return value
- See also
- Example

SalNumberAbs

Syntax	***nNum*** **= SalNumberAbs** *(nValue)*
Description	Computes a number's absolute value.
Parameters	nValue Number. The number whose absolute value you want.
Return Value	nNum is the absolute value of nValue.
Example	`Actions` `  Set nNum1 = SalNumberAbs ( -12 )`

SalNumberArcCos

Syntax	***nArcCos*** **= SalNumberArcCos** *(nValue)*
Description	Computes the arccosine of a value in the range 0 to 1. The value's domain is -1 to 1.
Parameters	nValue Number. The number whose arccosine you want.
Return Value	nArcCos is the arccosine of nValue. It is NUMBER_Null if nValue is less than -1 or greater than 1.
Example	`Actions` `  Set nNum1 = SalNumberArcCos ( 1 )`

SalNumberArcSin

Syntax	***nArcSin*** **= SalNumberArcSin** *(nValue)*
Description	Computes a value's arcsine. The value's domain is -1 to 1.
Parameters	nValue Number. The number whose arcsine you want.
Return Value	nArcSin is the arcsine of nValue. nArcSin is NUMBER_Null if nValue is less than -1 or greater than 1.

Example
```
Actions
                Set nNum1 = SalNumberArcSin ( 1 )
```

SalNumberArcTan

Syntax ***nArcTan =*** **SalNumberArcTan** *(nValue)*

Description Computes a value's arctangent.

Parameters nValue Number. The number whose arctangent you want.

Return Value nArcTan is the arctangent of nValue. nArcTan is in the range -1 to 1.

See Also SalNumberArcTan2

Example
```
Actions
                Set nNum1 = SalNumberArcTan ( 1 )
```

SalNumberArcTan2

Syntax ***nArcTan2 =*** **SalNumberArcTan2** *(nValueY, nValueX)*

Description Computes the arctangent of two values. This function uses the signs of both parameters to determine the quadrant of the return value.

Parameters nValueY Number. One of two values whose arctangent you want. nValueX Number. The other of two values whose arctangent you want.

Return Value nArcTan2 is the arctangent of nValueY and nValueX. nArcTan2 is in the range -1/2 to 1/2.

See Also SalNumberArcTan

Example
```
Actions
    Set nNum1 = SalNumberArcTan2 ( 1, 0 )
```

SalNumberCos

Syntax ***nCos =*** **SalNumberCos** *(nAngle)*

Description Computes an angle's cosine. You must specify the angle in terms of radians. Parameters nAngle Number. The value of the angle whose cosine you want. Return Value nCos is the cosine of nAngle. If the angle is large, nCos can reflect a partial loss of significance. If the angle is so large that significance is totally lost, SalNumberCos returns zero (0).

See Also SalNumberCosH

Example
```
Actions
    Set nNum1 = SalNumberCos ( 0 )
```

SalNumberCosH

Syntax ***nCosH =*** **SalNumberCosH** *(nAngle)*

Description Computes an angle's hyperbolic cosine. You must specify the angle in terms of radians.

Parameters nAngle Number. The value of the angle whose hyperbolic cosine you want.

Return Value nCosH is the hyperbolic cosine of nAngle. If the return value is too large, nCosH equals zero (0).

See Also SalNumberCos

Example
```
Actions
    Set nNum1 = SalNumberCosH ( 0 )
```

SalNumberExponent

Syntax ***nNumExp* = SalNumberExponent** *(nValue)*

Description Computes a value's exponential function.

Parameters nValue Number. The value whose exponential function you want.

Return Value nNumExp is the result of 'e' to the power of nValue. When there is an underflow or overflow, nNumExp is equal to NUMBER_Null.

Example
```
On Actions
    Set nNumExp = SalNumberExponent ( 2.302585093 )
```

SalNumberHigh

Syntax ***nHi* = SalNumberHigh** *(nValue)*

Description Returns a number's high-order word value (most significant 16 bits).

Parameters nValue Number. The number whose high-order word value you want.
TD Mobile treats nValue as an unsigned 32-bit number.

Return Value nHi is the high-order word value of nValue.

See Also SalNumberLow

VisNumberMakeLong (in Visual Toolchest section of online help)

Example
```
On Actions
    Set nHi = SalNumberHigh ( 0xffffaaaa )
```

SalNumberHypot

Syntax ***nHypotenuse* = SalNumberHypot** *(nX, nY)*

Description Computes the length of the hypotenuse of a right triangle, given the lengths of the other two sides.

Parameters nX Number. The length of one side of a right triangle.

nY Number. The length of another side of a right triangle.

Return Value nHypotenuse is the length of the hypotenuse of a right triangle. If the computation of the hypotenuse results in an overflow, nHypotenuse is equal to zero (0).

Example
```
Actions
            Set nHypotenuse = SalNumberHypot ( 3, 4 )
```

SalNumberLog

Syntax ***nLog* = SalNumberLog** *(nValue)*

Description Computes a number's natural logarithm.

Parameters nValue Number. The number whose natural logarithm you want.

Return Value nLog is the natural logarithm of nValue. If nValue is negative or 0, nLog is equal to NUMBER_Null.

See Also SalNumberLogBase10

Example
```
Actions
    Set nLog = SalNumberLog ( 1000 )
```

SalNumberLogBase10

Syntax ***nLogBase10 =*** **SalNumberLogBase10** *(nValue)*

Description Computes a number's base -10 logarithm.

Parameters nValue Number. The number whose base -10 logarithm you want.

Return Value nLogBase10 is the base-10 logarithm of nValue. If nValue is negative or 0, nLogBase10 is equal to NUMBER_Null.

See Also SalNumberLog

Example
```
Actions
    Set nLogBase10 = SalNumberLogBase10 ( 1000 )
```

SalNumberLow

Syntax ***nLo =*** **SalNumberLow** *(nValue)*

Description Returns a number's low-order word value (least significant 16 bits).

Parameters nValue Number. The number whose low-order word value you want.
TD Mobile treats nValue as an unsigned 32-bit number.

Return Value nLo is the low-order word value of nValue.

See Also SalNumberHigh

VisNumberMakeLong (in Visual Toolchest section of online help)

Example
```
Actions
    Set nLo = SalNumberLow ( 0xffffaaaa )
```

SalNumberMax

Syntax ***nNumMax =*** **SalNumberMax** *(nVal1, nVal2)*

Description Returns the greater of two values.

Parameters nVal1 Number. The first of two values. nVal2 Number.
The second of two values.

Return Value nNumMax is the greater of nVal1 and nVal2.

See Also SalNumberMin

Example
```
Actions
    Set nNumMax = SalNumberMax ( 1765.2, -2 )
```

SalNumberMin

Syntax ***nNumMin =*** **SalNumberMin** *(nVal1, nVal2)*

Description Returns the lesser of two values.

Parameters nVal1 Number. The first value. nVal2 Number.

The second value.

Return Value nNumMin is the lesser of nVal1 and nVal2.

See Also SalNumberMax

Example
```
Actions
    Set nNumMin = SalNumberMin ( 1765.2, -2 )
```

SalNumberMod

Syntax ***nModulo =*** **SalNumberMod** *(nNumber, nNumberMod)*

Description Returns a number's modulo. This function divides nNumber by nNumberMod and returns the remainder.

Return Value nModulo is the remainder of nNumber divided by nNumberMod.

Example
```
Actions
    Set nNumber = SalNumberMod ( 5, 2 )
```

SalNumberPi

Syntax ***nNumPi =*** **SalNumberPi** *(nValue)*

Description Multiples a number by Pi. Pi is equal to 3.1415926535979323. Parameters nValue

Number. The number to multiply by Pi. Return Value nNumPi is nValue multiplied by Pi.

Example
```
Actions
    Set nNumPi = SalNumberSin ( SalNumberPi ( 1 ) /2 )
```

SalNumberPower

Syntax ***nNumPower =*** **SalNumberPower** *(nX, nY)*

Description Computes nX raised to the power of nY. This function does not recognize integral, floating-point values greater than 2 to the 64th power, such as 1.0E100.

Parameters nX Number. The number to raise to the power of nY. nY Number. The exponent.

Return Value nNumPower is equal to nX raised to the nYth power, with the following conditions:

1. If nX is not 0 and nY is 0, nNumPower is equal to 1.
2. If nX is 0 and nY is negative, nNumPower is equal to zero (0).
3. If both nX and nY are zero (0), or if nX is negative and nY is not a whole number, nNumPower is equal to zero (0), meaning that an error occurred.
4. In instances where an overflow or an underflow occurs, nNumPower is equal to zero (0).

Example
```
Actions
            Set nNumPow = SalNumberPower ( 2, 3 )
```

SalNumberRandInit

Syntax ***bOk =*** **SalNumberRandInit** *(nSeed)*

Description Sets the starting point for generating a series of pseudo-random numbers using SalNumberRandom.

Use SalNumberRandInit when you want to generate the same set of pseudo-random numbers over and over again, for example, when doing reproducible experiments.

Call SalNumberRandInit followed by numerous calls to SalNumberRandom. To repeat the random number series, call SalNumberRandInit again, specify the same seed value, and follow with numerous calls to SalNumberRandom.

Parameters nSeed Number. The starting point. A whole number in the range of 0 to 32767.

Return Value bOk is TRUE if the function succeeds and FALSE if it fails.

See Also SalNumberRandom

Example

```
Actions
    Call SalNumberRandInit ( 12 )
```

SalNumberRandom

Syntax ***nRandomNum* = SalNumberRandom** *( )*

Description Generates a pseudo-random number. The numbers generated by this function are integers (whole numbers) in the range 0 to 32767 (0 to 0x7FFF).

Parameters No parameters.

Return Value nRandomNum is a pseudo-random number.

See Also SalNumberRandInit

Example

```
Actions
    Call SalNumberRandInit ( 12 )
    Set nRandNum = SalNumberRandom ( )
```

SalNumberRound

Syntax ***nResult* = SalNumberRound** *(nNumber)*

Description Returns a rounded number.

If the fractional part of a number is greater than or equal to .5, TD Mobile rounds the number up. For example, the number 124.33 returns 124; the number 124.56 returns 125.

Parameters nNumber Number. The number to round.

Return Value nResult is nNumber after rounding.

Example

```
Actions
    Set nNumber = SalNumberRound ( 124.5 )
```

SalNumberSin

Syntax ***nSin* = SalNumberSin** *(nAngle)*

Description Computes an angle's sine. You must specify the angle in terms of radians. Parameters nAngle Number. The value of the angle whose sine you want. Return Value nSin is the sine of nAngle. If the angle is large, nSin can reflect a partial loss of significance. If the angle is so large that significance is totally lost, nSin is equal to zero (0).

See Also SalNumberSinH

Example
```
Actions
    Set nNum = SalNumberSin ( SalNumberPi ( 1 ) /2 )
```

SalNumberSinH

Syntax ***nSinH =*** **SalNumberSinH** *(nAngle)*

Description Computes an angle's hyperbolic sine. You must specify the angle in terms of radians.

Parameters nAngle Number. The value of the angle whose hyperbolic sine you want.

Return Value nSinH is the hyperbolic sine of nAngle. If the angle is too large, nSinH is equal to zero (0).

See Also SalNumberSin

Example
```
Actions
    Set nNum = SalNumberSinH ( 0 )
```

SalNumberSqrt

Syntax ***nSqrt =*** **SalNumberSqrt** *(nValue)*

Description Computes a number's square root.

Parameters nValue Number. The number whose square root you want.

Return Value nSqrt is the square root of nValue. If nValue is negative, it is out of the domain of valid values and nSqrt is equal to zero (0).

Example
```
Actions
    Set nSqrt = SalNumberSqrt ( 36 )
```

SalNumberTan

Syntax ***nTan =*** **SalNumberTan** *(nAngle)*

Description Computes an angle's tangent. You must specify the angle in terms of radians. Parameters nAngle Number. The value of the angle whose tangent you want. Return Value nTan is the tangent of nAngle. If the angle is large, nTan can reflect a partial loss of significance. If the angle is so large that significance is totally lost, nTan is equal to zero (0).

See Also SalNumberTanH

Example
```
Actions
    Set nNum = SalNumberTan ( SalNumberPi ( 1 ) / 4 ) )
```

SalNumberTanH

Syntax ***nTanH =*** **SalNumberTanH** (nAngle)

Description Computes an angle's hyperbolic tangent. You must specify the angle in terms of radians.

Parameters nAngle Number. The value of the angle whose hyperbolic tangent you want.

Return Value nTanH is the hyperbolic tangent of nAngle. If the angle is large, nTanH can reflect a partial loss of significance.

If the angle is so large that significance is totally lost, nTanH is equal to zero (0).

See Also SalNumberTan

Example
```
Actions
    Set nNum = SalNumberTanH ( 0 )
```

SalNumberToChar

Syntax ***strChar* = SalNumberToChar** *(nNumber)* Description Converts a decimal value to an ASCII character. Parameters nNumber Number. The number to convert. Return Value strChar is the character converted from nNumber.

See Also SalStrFirstC

Example
```
Actions

    Set v2 = SalNumberToChar ( v1 )
```

SalNumberToHString

Syntax ***strString* = SalNumberToHString** *(nHString)*

Description Converts a number to a string handle.

Parameters nHString Number. The numeric value of the string handle to convert.

Return Value strString is a string handle that represents the number converted.

See Also SalHStringToNumber

Example
```
Actions
    Set strString = SalNumberToHString ( lParam )
                  Set nBuffLen = SalStrGetBufferLength (strString)
```

SalNumberToStr

Syntax ***nLength* = SalNumberToStr** *(nNumber, nDecimalPlaces, strString)*

Description Converts a number to a string.

Parameters nNumber Number. The number to convert.

nDecimalPlaces Number. The number of decimal places you want in strString. strString Receive String. The string converted from nNumber.

Return Value nLength is the length of strString, including the decimal point. strString is the string converted from nNumber.

See Also SalNumberToStrX SalStrToNumber

Example
```
Actions
    Set nLength = SalNumberToStr ( 124.5, 1, strString )
```

SalNumberToStrX

Syntax ***strString* = SalNumberToStrX** *(nNumber, nDecimalPlaces)*

Description Converts a number to a string.

Parameters

nNumber	Number. The number to convert.
nDecimalPlaces	Number. The number of decimal places you want in strString.
strString	Receive String. The string converted from nNumber.

Return Value nLength is the length of strString, including the decimal point. strString is the string converted from nNumber.

See Also SalNumberToStr
SalStrToNumber

Example

```
Actions
      Set var2 = SalNumberToStrX ( var1, 2 )
```

SalNumberTruncate

Syntax ***nResult* = SalNumberTruncate** *(nNumber, nPrecision, nScale)*

Description Truncates a number.

Parameters

nNumber	Number. The number to truncate, starting with the leftmost. nPrecision
	Number. The number of digits to display, starting with the leftmost.
nScale	Number. The number of digits to the right of the decimal point. The nPrecision parameter must be large enough to hold the number of digits that you specify in this parameter.

Return Value nResult is the result of truncating nNumber.

Example

```
Actions
      Set nNum = SalNumberTruncate( 10.0625, 4, 4
```

Object Functions

This is an alphabetical list of the SAL object functions accompanied by detailed information about each function's purpose, its parameters and return value, and an example.

Function descriptions include:

- Syntax
- Description
- Parameters
- Return value
- See also
- Example

SalObjCreateFromString

Syntax ***RefObject* = SalObjCreateFromString**(*StrClassName*)

Description Creates an object of a user-defined class. The class name will be determined by the value of StrClassName.

Parameters: StrClassName String. The name of the user-defined class.

Return Value Reference. Reference to the object created if the function succeeds, OBJ_Null if the function fails.

Example

```
Class Definitions


  Internal Functions Function:
GetAnimalObj Description:
Returns: CAnimal
Parameters: Static Variables Local
Variables Actions
  Return SalObjCreateFromString( "CAnimal" )
```

See Also SalObjIsDerived() SalObjGetType() SalObjIsNull() SalObjIsValidClassName()

SalObjGetType

Syntax ***StrClassName = SalObjGetType***(*RefObject*)

Description Determine the class name of the object referred to by RefObject. This function call returns the actual type of the object referred to by RefObject, not the declared type of RefObject.

Parameters RefObject Reference. Reference to the object whose class name is to be determined.

Return Value String. Class name of the object referred to by RefObject if successful, STRING_Null if unsuccessful.

Example

```
Call SalObjGetType( RefAnimal )
```

See Also SalObjCreateFromString() SalObjIsDerived() SalObjIsNull() SalObjIsValidClassName()

SalObjIsDerived

Syntax ***bDerived =*** **SalObjIsDerived***(RefObject, StrClassName)*

Description Determine if the object referred to by RefObject is an instance a certain user-defined class, StrClassName, or an instance of a subclass inherited from the user-defined class.

Parameters RefObject Reference. Reference to the object in question.

StrClassName String. The name of the class to use for the match.

Return Value TRUE if RefObject is an instance of the class identified by the value of StrClassName or if RefObject is an instance of a subclass of the class identified by the value of StrClassName; FALSE otherwise.

Example

```
Functional Class: CAnimal
  Derived From:

Functional Class: CDog
  Derived From: CAnimal

Local Variable: CAnimal: RefAnimal
  CDog:    Ref Dog

Actions
  If ( SalObjIsDerived( RefAnimal), "CAnimal")
    Set bAnimal = TRUE

  If( SalObjIsDerived( RefDog ), "CAnimal" )
    Set bDog = TRUE
```

See Also SalObjCreateFromString SalObjGetType SalObjIsNull SalObjIsValidClassName

SalObjIsValidClassName

Syntax ***bValidClassName =*** **SalObjIsValidClassName***(StrClassName)*

Determine whether StrClassName holds a valid user-defined class name.

Parameters StrClassName String. Name of the class to check.

Return Value TRUE if StrClassName holds a valid user-defined class name, FALSE otherwise.

Example

```
Functional Class: CAnimal
  Derived From:

Functional Class: CDog
  Derived From: CAnimal

Local Variable: CAnimal: RefAnimal

Actions:

  If ( SalObjIsValidClassName( "CDog" ) AND SalObjIsDerived(
         RefAnimal, "CDog" )
    Set RefAnimal = SalObjCreateFromString( "CDog" )
```

See Also SalObjCreateFromString SalObjGetType
SalObjIsDerived SalObjIsNull

```
et hWndParent = SalParentWindow ( hWnd )
```

SQL Functions

This is an alphabetical list of the SAL SQL functions accompanied by detailed information about each function's purpose, its parameters and return value, and an example.

Function descriptions include:

- Syntax
- Description
- Parameters
- Return value
- See also
- Example

SqlClearImmediate

Syntax	***bOk* = SqlClearImmediate** *( )*
Description	Disconnects the internal Sql Handle from a database. You connect the internal handle to a database by calling SqlImmediate and it remains connected until the application terminates or you explicitly disconnect it with SqlClearImmediate. SqlClearImmediate causes an implicit COMMIT if it is the last cursor you disconnect from the database.
Parameters	No parameters.
Return Value	bOk is TRUE if the function succeeds and FALSE if it fails.
See Also	SqlImmediate
Example	`Set bOk = SqlClearImmediate ( )`

SqlClose

Syntax ***bOk =* SqlClose** *(hSql)*

Description Invalidates a SQL command and/or frees the cursor name associated with the specified cursor, making it available for re-use.

If you create a named cursor by calling SqlOpen and then instead of closing it, call SqlOpen or SqlExecute again, you get an error that the name has already been used. Parameters hSql Sql Handle. A handle that identifies a database connection. Return Value bOk is TRUE if the function succeeds and FALSE if it fails.

See Also SqlOpen

Example
```
Set bOk = SqlClose ( hSql )
```

SqlCommit

Syntax ***bOk =* SqlCommit** *(hSql)*

Description Commits all of the SQL transaction's cursors that are connected to the same database.

NOTE: To prevent destroying a cursor's result set when a COMMIT is performed, turn on cursor context preservation by calling SqlSetParameter and setting the DBP_PRESERVE parameter to TRUE.

Parameters hSql Sql Handle. A handle that identifies a database connection.

Return Value bOk is TRUE if the function succeeds and FALSE if it fails.

Example
```
Call SqlCommit ( hSql )
```

SqlConnect

Syntax ***bOk =* SqlConnect** *(hSql)*

Description Connects to a database. The connection will be via OLE DB or via native routers, depending on the presence or absence of a value in system variable SqlUDL.

TD Mobile uses the values in the SqlUDL, SqlDatabase, SqlUser, and SqlPassword variables. The default values for these variables are (none), DEMO, SYSADM, and SYSADM. The value of other system variables such as SqlNoRecovery, SqlInMessage, and SqlOutMessage take effect after this function executes.

SqlUDL is a system variable that can contain a provider name, a connection string, or the name of a UDL file to use for OLE DB connection information. This variable was introduced in version 3.1. One of its purposes is to ease the migration of existing TD Mobile applications from use of native routers to use of OLE DB. In many cases, existing apps simply need a few lines to set the value of SqlUDL and the rest of the app will run smoothly against OLE DB

To accomplish this, function SQLConnect has been altered in TD Mobile version 3.1. SQLConnect now looks first at variable SqlUDL and, if it finds a file name in that variable, reads connection information from that file. If it finds a provider name or connection string in SqlUDL, it uses the provider name. *However, variables SqlDatabase, SqlUser and SqlPassword may still affect the connection information*. If the database name or user name or password was not specified from the SqlUDL information, SQLConnect will obtain the needed value from those three variables. If the SqlUDL information was complete, but there is also a value in SqlDatabase, SqlUser, or SqlPassword, that value will *override* whatever had been in the connection information. This function then forms a connection string from that information, then makes an OLE DB connection with that string.

Because variable SqlPassword can override any password information that may have been in the connection string, you can keep password information out of the UDL file and supply it programmatically at runtime instead, for greater security.

If SqlUDL is null, SqlConnect uses the older (API and routers) method of connecting with the values of SqlDatabase, SqlUser, and SqlPassword..

Parameters hSql Receive Sql Handle. A handle that identifies a database connection.

Return Value bOk is TRUE if the function succeeds and FALSE if it fails.

See Also SqlDisconnect

Example
```
Call SqlConnect ( hSqlPrimary )
```

SqlContextClear

Deprecated. This function has been deprecated and should no longer be used. Instead use SqlVarSetup.

Syntax ***bOk =* SqlContextClear** *(hSql)*

Description Clears the context set by SqlContextSet or SqlContextSetToForm. TD Mobile evaluates the bind and into variables associated with the specified Sql Handle in the local context.

Parameters hSql Sql Handle. A handle that identifies a database connection.

Return Value bOk is TRUE if the function succeeds and FALSE if it fails.

See Also SqlContextSet SqlContextSetToForm
SqlImmediateContext

Example
```
Set bOk = SqlContextClear ( hSql )
```

SqlContextSet

Deprecated. This function has been deprecated and should no longer be used. Instead use SqlVarSetup.

Syntax ***bOk =* SqlContextSet** *(hSql)*

Description Sets the context for future processing (for example, calls to SqlPrepare, SqlFetchNext, SqlFetchPrevious, and SqlFetchRow). Sql* functions you call after SqlContextSet behave as if they are in the window identified by hWndForm.

Call this function in a class to perform SQL processing for the current window without fully qualifying bind and into variables. This function is also useful for global functions.

~~**Important:** After you call SqlContextSet, the context for bind variables and into variables is~~ **always** hWndForm. If you call a Sql* function in an internal function, window function, or class function after calling SqlContextSet, TD Mobile does not recognize local variables or parameters that you use as bind variables and into variables.

Parameters hSql Sql Handle. A handle that identifies a database connection.

Return Value bOk is TRUE if the function succeeds and FALSE if it fails.

See Also SqlContextClear SqlContextSetToForm
SqlImmediateContext

Example
```
Set bOk = SqlContextSet ( hSql )
```

SqlDirectoryByName

Syntax ***bOk =* SqlDirectoryByName** *(strServerName, strArrayNames)*

Description Returns the database names on the specified server.

Parameters strServerName String. The name of a server.

	strArrayNames	String Array. The name of an array of strings containing database names.

Return Value bOk is TRUE if the function succeeds and FALSE if it fails.

Example
```
Actions
   Call SqlDirectoryByName ( 'server1', strDatabaseNames )
```

SqlDisconnect

Syntax ***bOk =*** **SqlDisconnect** *(hSql)*

Description Disconnects from a database.

Disconnecting the last Sql Handle from a database causes an implicit COMMIT of the database. Disconnect all Sql Handles before the application exits.

Parameters	hSql	Sql Handle. The handle that identifies the database connection to disconnect.

Return Value bOk is TRUE if the function succeeds and FALSE if it fails.

See Also SqlConnect

Example
```
Actions
      …
      Call SqlDisconnect ( hSqlPrimary )
```

SqlError

Syntax ***nError =*** **SqlError** *(hSql)*

Description Returns the most recent error code for the specified Sql Handle.

SqlError is not useful after a call to SqlImmediate because SqlImmediate does not return a handle that you can use as the parameter for SqlError.

Parameters	hSql	Sql Handle. The handle on which an error occurred.

Return Value nError is the error code returned. It is equal to zero (0) if no error occurred.

See Also SqlExtractArgs

Example
```
Set nSqlError = SqlError ( hSqlPrimary )
```

SqlErrorText

Syntax ***bOk =* SqlErrorText** *(nError, nType, strError, nLength, nRealLength)*

Description Gets the error reason or remedy for the specified error code from ERROR.SQL.
Call SqlError to get the most recent error code. When your application detects an error condition, you can use the error code returned by SqlError to look up the error reason and remedy with SqlErrorText.

When connected to an OLE DB data source, do not use this function; use SqlGetSessionErrorInfo instead.

Parameters nError Number. A SQLBase error code.

nType Number. Specify one or both (by combining them with the OR (|) operator) of these constants:

Constant	Description
SQLERROR_Reason	Retrieve error code reason.
SQLERROR_Remedy	Retrieve error message remedy.

strError Receive String. The reason or remedy explanation. nLength

Number. The maximum length of strError. nRealLength Receive Number. The actual length

of strError.

Return Value bOk is TRUE if the function succeeds and FALSE if it fails.

See Also SqlError SqlGetErrorText SqlGetErrorTextX

Example
```
Set bOk = SqlErrorText ( nError, nType, strError, nLength, nRealLength )
```

SqlExecute

Syntax ***bOk =* SqlExecute** *(hSql)*

Description Executes a SQL statement that was prepared with SqlPrepare or retrieved with SqlRetrieve.

SqlExecute does not fetch data. To fetch data, call one of the SqlFetch* functions: SqlFetchNext, SqlFetchPrevious, or SqlFetchRow.

Bind variables are sent to the database when you call SqlExecute.

You can use SqlExecute just like SqlOpen, but you can never address rows in the result set by a cursor name. That is, you cannot use the 'CURRENT OF <cursor_name>' and 'ADJUSTING <cursor_name>' clauses to UPDATE, DELETE or INSERT result set rows.

Parameters hSql Sql Handle. The handle associated with a SQL statement.

Return Value bOk is TRUE if the function succeeds and FALSE if it fails.

See Also SqlOpen

Example
```
Call SqlExecute ( hSqlPrimary )
```

SqlExecutionPlan

Syntax ***bOk =* SqlExecutionPlan** *(hSql, strString, nLength)*

Description Gets the execution plan for a compiled SQL statement. An execution plan shows the tables, views, indexes, and optimizations for the SQL statement. Tables and views are listed in the order in which they are processed.

Parameters

hSql	Sql Handle. The handle associated with a compiled SQL statement.
strString	String. The execution plan. Also, a Receive parameter.
nLength	Number. The maximum length of the execution plan. Also, a Receive parameter.

Return Value bOk is TRUE if the function succeeds and FALSE if it fails.

Example
```
Set bOk = SqlExecutionPlan ( hSql, strString, nLength )
```

SqlExists

Syntax ***bOk =*** **SqlExists** *(strSelect, bExists)*

Description Determines whether a row exists.

SqlExists uses the values of the SqlDatabase, SqlUser, and SqlPassword variables to connect to a database, and uses an internal Sql Handle to execute the specified query.

Parameters

strSelect	String. The SELECT statement that establishes the existence of a row.
bExists	Receive Boolean. TRUE if the row exists and FALSE if it does not.

Return Value bOk is TRUE if strSelect is correct and executable and FALSE otherwise.

Example
```
Call SqlExists ( 'SELECT * FROM ' || strTable ||
    'WHERE ' || strExistsColumn || ' = ' || '\'' ||
      strExistsObject || '\'', bExists )
```

SqlExtractArgs

Syntax ***bOk =*** **SqlExtractArgs** *(wParam, lParam, hSql, nError, nPos)*

Description Extracts information from the SAM_SqlError wParam and lParam arguments. Call this function only while processing a SAM_SqlError message which is sent when an error occurs while executing a SQL function.

Parameters

wParam	Number. The value of the wParam argument of the SAM_SqlError message.
lParam	Number. The value of the lParam argument of the SAM_SqlError message.
hSql	Receive Sql Handle. The handle of the function that got an error. TD Mobile extracts this value from the wParam argument.
nError	Receive Number. The error code. TD Mobile extracts this value from the low-order word of the lParam argument.
nPos	Receive number. The error position, if relevant to the function call. TD Mobile extracts this value from the high-order word of the lParam argument.

Return Value bOk is TRUE if the function succeeds and FALSE if it fails.

See Also SqlError
SqlGetErrorPosition

Example
```
Call SqlExtractArgs ( wParam, lParam, hSqlError, nSqlError, nErrorPos )
```

SqlFetchNext

Syntax **_bOk_ = SqlFetchNext** *(hSql, nInd)*

Description Fetches the next row in a result set. You must have prepared the SELECT statement with SqlPrepare and executed it with SqlExecute, or opened it with SqlOpen.

Parameters hSql Sql Handle. The handle of a SELECT statement.

nInd Receive Number. The fetch return code is one of the FETCH_* values.

Return Value bOk is TRUE if another row was fetched and FALSE if no row was fetched.
SqlFetchNext does not return FALSE and Fetch indicator does not show EOF until you attempt to fetch past the last row.

See Also SqlFetchPrevious
SqlFetchRow

Example
```
Call SqlFetchNext ( hSqlPrimary, nRetVal )
```

SqlFetchPrevious

Syntax **_bOk_ = SqlFetchPrevious** *(hSql, nInd)*

Description Fetches the previous row in a result set. You must have prepared the SELECT statement with SqlPrepare and executed it with SqlExecute, or opened it with SqlOpen.

Parameters hSql Sql Handle. The handle of a SELECT statement. nInd Receive Number. The fetch return code is one of the FETCH_* values.

Return Value bOk is TRUE if there is another row to fetch and FALSE otherwise.

See Also SqlFetchNext
SqlFetchRow

Example
```
Call SqlFetchPrevious ( hSqlPrimary, nRetVal)
```

SqlFetchRow

Syntax **_bOk_ = SqlFetchRow** *(hSql, nRow, nInd)*

Description Fetches a row according to an absolute row position. You must have prepared the SELECT statement with SqlPrepare and executed it with SqlExecute, or opened it with SqlOpen.

Note that when connected to an Oracle database, you must first set SqlResultSet to FALSE before calling SqlFetchRow. When connected to SQLBase or any non-Oracle database, you must first set SqlResultSet to TRUE before calling SqlFetchRow.

Parameters hSql Sql Handle. The handle of a SELECT statement. nRoNumber. The row number of the row to fetch.

nInd Receive Number. The fetch return code is one of the FETCH_* values.

Return Value bOk is TRUE if nRow could be fetched and FALSE otherwise.

See Also SqlFetchNext
SqlFetchPrevious

Example
```
Call SqlFetchRow ( hSqlPrimary, lParam, nRetVal )
```

SqlGetCommandText

Syntax **_bOk_ = SqlGetCommandText** (*hSql, sText*)

Description This function returns the SQL command last prepared on the specified SQL handle.
This function is only supported against OLE DB connections. If the Sql Handle was not created with an OLE DB provider, then, the function returns FALSE. If the call is made before a SQL command was prepared (either by SqlPrepare, SqlPrepareAndExecute or SqlPrepareSP), then the function returns FALSE.

Note that the function returns the SQL string even if the prepare failed (due to wrong syntax etc).

There is a related function named SqlGetLastStatement. That function doesn't take a Sql Handle as a parameter and returns the last statement that was prepared in the entire application. With OLE DB applications, Gupta does not recommend using that function.

Parameters

hSql	Sql Handle. The Sql handle associated with the desired statement handle.
sText	Receive String. The text of the command that was prepared.

Return Value

bOk	Boolean. TRUE for success and FALSE for failure.

SqlGetCursor

Syntax *nCursorHandle* = SqlGetCursor (*hSql*)

Description This function gets the actual cursor handle associated with a SQL handle. The cursor handle returned is useful when calling functions in the SQLBase API. It is equivalent to the second parameter of SQLBase API function *sqlcnc*.

This function is only valid with SQLBase and native routers. Do not use it against an OLE DB connection.

Parameters

hSql	Sql Handle. The logical SQL handle for which you seek the statement handle.

Return Value

nCursorHandle	Number. Use this cursor handle when calling a large number of SQLBase API functions that require such a handle.

SqlGetError

Syntax ***bOk =* SqlGetError***(hSql, nError, strErrorString)*

Description Turns off backend error mapping and reports real backend errors.

If the error number is less than 20,000, the file `ERROR.SQL` is searched for the error text and that text (if found) is returned; otherwise, the translated error number and database error message from the database server are returned.

When connected to an OLE DB data source, do not use this function; use SqlGetSessionErrorInfo instead.

Parameters

hSql	Sql Handle. The handle of a SELECT statement. nError Number. The error number.
strErrorString	String. The error text.

Return Value bOk is TRUE if the function succeeds and FALSE if it fails.

You specify the Sql Handle in hSql and SqlGetError returns the last error number and error text in nError and strErrorString.

If the backend is SQLBase, this function does the same thing as calling SqlError and SqlGetErrorText.

Example

```
Set bOk = SqlGetError( hSql, nError, strErrorString )
```

SqlGetErrorPosition

Syntax ***bOk =*** **SqlGetErrorPosition** *(hSql, nPos)*

Description Returns the offset of the error position within a SQL statement. After a SqlPrepare, the error position points to the place in the SQL statement where TD Mobile detected aSyntax error. The first character position in the SQL statement is zero (0).

This function is valid only when used against a SQLBase database connection. When used with any other database router, including OLE DB, the function always returns zero.

Parameters hSql Sql Handle. The handle of a SELECT statement.

nPos Receive Number. The position in the SQL statement where aSyntax error occurred.

Return Value bOk is TRUE if the function succeeds and FALSE if it fails.

See Also SqlExtractArgs

Example

```
Call SqlGetErrorPosition ( hSqlPrimary, nErrorPos )
```

SqlGetErrorText

Syntax ***bOk =*** **SqlGetErrorText** *(nError, strText)*

or

strText = **SqlGetErrorTextX** *(nError)*

Description Gets the message text for a SQL error number from ERROR.SQL.

When connected to an OLE DB data source, do not use this function; use SqlGetSessionErrorInfo instead.

Parameters nError Number. The error number. strText Receive

String. The error text.

Return Value bOk is TRUE if the function succeeds and FALSE if it fails. strText is the message text for

nError.

See Also SqlErrorText
SqlExtractArgs

Example

```
Call SqlGetErrorText ( nError, strText )

or

Set strText = SqlGetErrorTextX ( nError )
```

SqlGetErrorTextX

Syntax ***strText =*** **SqlGetErrorTextX** (*nError*)

Description When the user chooses the Insert menu item, this example compiles a SQL statement for execution. To process any invalid SQL statements and trap the error (bypassing the default error processing), add the "When SqlError" statement with a FALSE return before the SqlPrepare. When SqlPrepare returns FALSE, call SqlError to get the error number, call SqlGetErrorTextX to get the error description, and call SqlGetErrorPosition to get the character position where theSyntax error was detected.

Example

```
Set strErrorText = SqlGetErrorTextX ( nSqlError )
```

SqlGetLastStatement

Syntax ***sSqlStatement* = SqlGetLastStatement*( )***

Description Returns the last SQL statement passed to a SqlXxxx function for any cursor. The statement returned is the same statement that would be shown in the default SQL Error dialog box.

Note: The statement is global for all cursors, therefore, if you get a SQL error after another cursor has had a statement prepared the statement returned may not be the one prepared for the handle.

Parameters No parameters.

Return Value sSqlStatement contains the last SQL statement.

Example
```
When SqlError
                Set sStatement = SqlGetLastStatement()
```

SqlGetModifiedRows

Syntax ***bOk* = SqlGetModifiedRows** *(hSql, nCount)*

Description Returns the number of rows affected by the most recent INSERT, UPDATE, or DELETE statement.

Parameters hSql Sql Handle. The handle of a SQL statement. nCount Receive Number.

The number of rows affected.

Return Value bOk is TRUE if the function succeeds and FALSE if it fails.

Example
```
Set bOk = SqlGetModifiedRows ( hSql, nCount )
```

SqlGetParameter

Syntax ***bOk* = SqlGetParameter** *(hSql, nParameter, nNumber, strString)*

Description Gets the value of a database parameter. This function returns the parameter value in nNumber or strString as appropriate for the data type of the parameter.

When using a connection to database servers other than SQLBase you *cannot* manipulate parameters that are *specific* to those databases with SqlGetParameter. You must use SqlGetParameterAll instead.

Parameters
- hSql — Sql Handle. A handle that identifies a database connection.
- nParameter — Number. The database parameter. Specify one of the DBP_* constants.
- nNumber — Receive number. The value (TRUE or FALSE) of the parameter.

  If nParameter is DBP_BRAND, nNumber is one of the DBV_BRAND_* values.
- strString — Receive string. If you specify DBP_VERSION in nParameter, this is the version number.

Return Value bOk is TRUE if the function succeeds and FALSE if it fails.

See Also SqlGetParameterAll SqlSetParameter SqlSetParameterAll

Example
```
Actions
   Call SqlGetParameter ( hSqlPrimary, DBP_LOCKWAITTIMEOUT, nTimeout, strNull
      )
```

SqlGetParameterAll

Syntax ***bOk =* SqlGetParameterAll** *(hSql, nParameter, nNumber, strString, bNumber)*

Description Gets the value of a database parameter identified by a SQLP* constant value defined in SQL.H. This function returns the parameter value in nNumber or strString as appropriate for the data type of the parameter.

Important: A set of the SQLP* constants in SQL.H have the same values as the DBP_* constants, but the values identify different parameters. Be sure to specify the correct number.

Parameters

hSql	Sql Handle. A handle that identifies a database connection.
nParamete	Number. The database parameter. Specify the value of one the SQLP* constants defined in SQL.H.
nNumber	Receive number. The value of nparameter if it is a number.
	Receive string. The value of nParameter if it is a string.
bNumber	Boolean. If TRUE, the parameter value is returned in nNumber. If FALSE, the parameter value is returned in strString.

Return Value bOk is TRUE if the function succeeds and FALSE if it fails.

See Also SqlGetParameter SqlSetParameter
SqlSetParameterAll

Example
```
Set bOk = SqlGetParameterAll ( hSql, nParameter, nNumber, strString, bNumber
    )
```

SqlGetResultSetCount

Syntax ***bOk =* SqlGetResultSetCount** *(hSql, nCount)*

Description Counts the rows in a result set by building the result set. TD Mobile fetches each row that has not already been fetched, returns a count of the rows, and positions the cursor back to its original position. Warning: this can be time-consuming if the result set is large.

INSERTs into the result set increase the result set row count, but DELETEs — which display as blank rows in result set mode — do not decrease the row count. However, the deleted blank rows disappear on the next SELECT.

You must be in Result Set mode.

You must call SqlExecute before SqlGetResultSetCount.

Parameters hSql Sql Handle. A handle associated with a result set. nCount Receive Number. The number of rows in the result set.

Return Value bOk is TRUE if the function succeeds and FALSE if it fails.

Example
```
Actions
     Call SqlPrepare ( hSqlPrimary, strSqlTblWindow ) Call SqlExecute (
     hSqlPrimary )
     Call SqlGetResultSetCount ( hSqlPrimary, nRowCount )
```

SqlGetRollbackFlag

Syntax ***bOk =* SqlGetRollbackFlag** *(hSql, bRollbackFlag)*

Description Returns the database rollback flag. Use this function after an error to find out if a transaction rolled back.

TD Mobile sets the rollback flag when a system-initiated rollback occurs as the result of a deadlock or system failure. TD Mobile does not set the rollback flag on a user-initiated rollback.

This function is valid for connections that use native routers, but not for OLE DB connections.

Parameters	hSql	Sql Handle. The handle associated with the function call that got an error.
	bRollbackFlag	Receive Boolean. TRUE if a rollback occurred and FALSE otherwise.
Return Value	bOk is TRUE if the function succeeds and FALSE if it fails.	

Example

```
Call SqlGetRollbackFlag ( hSqlError, bRollbackFlag ) If bRollbackFlag
        ! Execute code to handle rolled back
                  ! transaction
```

SqlPrepare

Syntax ***bOk* = SqlPrepare** *(hSql, strSqlStatement)*

Description Compiles a SQL statement for execution. Compiling includes:

- Checking theSyntax of the SQL statement.
- Checking the system catalog.
- Processing a SELECT statement's INTO clause.

  An INTO clause names where data is placed when it is fetched. These variables are sometimes called INTO variables. You can specify up to 255
  INTO variables per SQL statement.
- Identifying bind variables in the SQL statement. Bind variables contain input data for the statement. You can specify up to 2558 bind variables per SQL statement.

Follow this function with a SqlOpen, SqlExecute, SalTblDoInserts, SalTblDoUpdates, or SalTblDoDeletes, or fetches.

Parameters hSql Sql Handle. A handle that identifies a database connection. strSqlStatement

String. The SQL statement to compile.

Return Value bOk is TRUE if the function succeeds and FALSE if it fails.

See Also SqlExecute

Example

```
Call SqlPrepare ( hSqlPrimary, 'INSERT INTO CUSTOMER '
                || ' ( CUSTOMER ) VALUES ' ||'(
              :frmCustomer.dfCustomer )' )
```

SqlPrepareAndExecute

Syntax ***bOk* = SqlPrepareAndExecute** *(hSql, strSqlStatement)*

Description Compiles and executes a SQL statement. Compiling includes:

- Checking the Syntax of the SQL statement.
- Checking the system catalog.
- Processing a SELECT statement's INTO clause.

An INTO clause names where data is placed when it is fetched. These variables are sometimes called INTO variables. You can specify up to 128
INTO variables per SQL statement.

- Identifying bind variables in the SQL statement. Bind variables contain input data for the statement. You can specify up to 128 bind variables per SQL statement.

Parameters

hSql	Sql Handle. A handle that identifies a database connection. strSqlStatement

String. The SQL statement to compile and execute.

Return Value bOk is TRUE if the function succeeds and FALSE if it fails.

See Also
SqlExecute
SqlPrepare

Example

```
Set bOk = SqlPrepareAndExecute ( hSql, 'Select name from' || ' employees into
                :df1' )
If bOk
     Call SqlFetchNext ( hSql, nInd )
```

SqlRetrieve

Syntax ***bOk =* SqlRetrieve** *(hSql, strName, strBindList, strIntoList)*

Description Retrieves a SQLBase compiled command.

To execute the command, you need only call SqlExecute. You do not need to compile the command with SqlPrepare because the command is compiled when it is stored with SqlStore.

Parameters

hSql	Sql Handle. A handle that identifies a database
strName	connection. String. The name of the compiled command.
strBindList	String. A comma-separated list of up to 128 TD Mobile bind variables. This list has the same number of variables as the compiled command. This string can be null.
strIntoList	String. A comma-separated list of up to 128 TD Mobile INTO variables. This list has the same (or less) number of INTO variables as named in the SELECT list of the compiled command. This string can be null (''), and should be null if the next command being executed is SalListPopulate.

Return Value bOk is TRUE if the function succeeds and FALSE if it fails.

See Also SqlDropStoredCmd

SqlStore

Example

```
Call SqlRetrieve ( hSqlPrimary, 'PRODUCTS', ':nPrice',
                              ':strLBItem' )
```

SqlSetInMessage

Syntax ***bOk =* SqlSetInMessage** *(hSql, nSize)*

Description Sets the size (in bytes) of the input message buffer for the specified Sql Handle. The input message buffer holds input for the application (such as the result of a query).

There is one input message buffer per connected Sql Handle on the client computer. The database server (or gateway) maintains one input message buffer that is the size of the largest input message buffer on the client

computer.

When fetching data, as many rows as possible are compacted into one input message buffer. Each FETCH reads the next row from the input message buffer until they are exhausted. At this point, if you are accessing a SQLBase database, SQLBase transparently fetches the next input buffer of rows depending on the isolation level.

A large input message buffer can improve performance because it reduces the number of network messages between the client and server. A large input message buffer can have a negative impact on concurrency, however, because any row currently in the buffer can have a shared lock on it (depending on the isolation level) which prevents other users from changing that row.

Parameters hSql Sql Handle. A handle that identifies a database connection. nSize Number. The size (in bytes) of the input message buffer that you want to set. The default is 1 Kbyte and the maximum is 32 Kbtyes.

Return Value bOk is TRUE if the function succeeds and FALSE if it fails.

See Also SqlSetOutMessage

Example

```
Actions
            Call SqlConnect ( hSqlPrimary )
     Call SqlSetInMessage ( hSqlPrimary, 10000 )
```

SqlSetIsolationLevel

Syntax ***bOk =* SqlSetIsolationLevel** *(hSql, strIsolation)*

Description Sets SQLBase's isolation level for all the application's cursors.

Read Repeatability is the default setting for TD Mobile.

Changing isolation levels causes an implicit commit of the database. The isolation level you set applies to all the Sql Handles for that user name that the application connects to the database. Read Committed (RC) behaves similarly to a Read Only (RO) transaction, with some differences:

A Read Only (RO) transaction views all the data that is selected as of the time the transaction started, while a Read Committed (RC) uses a "start of oldest active cursor" timestamp.

A Read Only (RO) transaction cannot modify any data, whereas a Read Committed (RC) transaction can.

Read Committed (RC) transactions behave like Release Locks (RL) isolation when modifying data. The RC1, RC2, and RC3 modes determine when the timestamp slides forward to the current time. How frequently it moves affects the consistency of the data returned for the SELECT. RC1 provides the most consistent data and RC3 provides the most "live" (least consistent) data. In all cases, the data is committed data combined with the current transaction uncommitted data.

Parameters

hSql	Sql Handle. A handle that identifies a database connection.	
strIsolation	String. The isolation level to set. Specify one of these values:	
	CS	Cursor Stability
	RL	Release Locks
	RO	Read Only
	RR	Read Repeatability

RC	Read Commited
RC1	Read Commited 1
RC2	Read Commited 2
RC3	Read Commited 3

Return Value bOk is TRUE if the function succeeds and FALSE if it fails.

Example
```
Actions
    Call SqlSetIsolationLevel ( hSqlPrimary, 'RL' )
```

SqlSetLockTimeout

Syntax ***bOk =* SqlSetLockTimeout** *(hSql, nTimeout)*

Description Specifies the maximum time to wait to acquire a lock. After the specified time elapses, a timeout occurs and the transaction rolls back.

Parameters

hSql	Sql Handle. A handle that identifies a database connection; the cursor on which you want to set a lock timeout value.
nTimeout	Number. The timeout period in seconds. Valid values include -1 (wait forever), 0 (never wait), and values up to and including 1800 (30 minutes). The default is 300.

Return Value bOk is TRUE if the function succeeds and FALSE if it fails.

Example
```
Actions
            Call SqlSetLockTimeout ( hSqlPrimary, 10 )
```

SqlSetLongBindDatatype

Syntax ***bOk =* SqlSetLongBindDatatype***(nBindVarNumber, nDatatype)*

Description Sets the type of column (text or binary) that a Long String binds to. By default, TD Mobile binds Long Strings to text columns. However, when you write or update a long binary column, call SqlSetLongBindDatatype and set the nDatatype parameter to binary (value = 23). Later you can set the type back to text by calling this function and setting nDatatype to text (value = 22). Value = 24 is for Informix and Ingres specific routers.

Call this function before executing the SQL statement (implicitly or explicitly) because TD Mobile compiles bind variables at execute time.

Use this function until Gupta Technologies LLC implements a native SAL binary long data type.

Parameters

nBindVarNumber	Number. The bind variable to set. The first bind variable in the SQL statement is 1, the second is 2, and so on. nDatatype

Number. The data type:

22 = text

23 = binary

24 = char \ long varchar > 254

Return Value bOk is TRUE if this function succeeds and FALSE if it fails.

Example
```
Call SqlSetLongBindDatatype( 1, BIND_Binary )
```

Note:

BIND_Text = 22

BIND_Binary = 23

BIND_INFORMIX_LText = 24

SqlSetOutMessage

Syntax ***bOk =* SqlSetOutMessage** *(hSql, nSize)*

Description Sets the size (in bytes) of the output message buffer for a specified Sql Handle. The output message buffer holds output from the application (such as a SQL command to compile or rows of data to insert into a database).

There is an output message buffer for each connected Sql Handle on the client computer. At the same time, the database server (or gateway) maintains an output message buffer that is the size of the largest of its clients' output message buffers.

A large output message buffer does not necessarily improve performance because the buffer only needs to be large enough to hold the largest SQL command to compile or the largest row of data to insert. (Rows are always sent to the database and inserted individually.) A large output message buffer can allocate space unnecessarily on both the client and the server, and it does not reduce network traffic.

Parameters hSql Sql Handle. A handle that identifies a database connection. nSize Number. The size (in bytes) of the output message buffer. The default is 1 Kbyte and the maximum is 32 Kbytes.

Return Value bOk is TRUE if the function succeeds and FALSE if it fails.

See Also SqlSetInMessage

Example
```
Actions
     Call SqlConnect ( hSqlPrimary )
     Call SqlSetOutMessage ( hSqlPrimary, 1500 )
```

SqlSetParameter

Syntax ***bOk =* SqlSetParameter** *(hSql, nParameter, nNumber, strString)*

Description Sets the value of a database parameter. Use the number (nNumber) and string (strString) arguments as appropriate for the data type of the parameter.

When using a connection to database servers other than SQLBase you *cannot* manipulate parameters that are *specific* to those databases with SqlSetParameter. You must use SqlSetParameterAll instead.

Parameters hSql Sql Handle. A handle that identifies a database connection.
nParameter Number. The database parameter to set. Specify one of the DBP_* constants.
nNumber Number. The value to assign to nParameter. Specify TRUE or FALSE for all but DBP_LOCKWAITTIMEOUT, for which you must specify a value in seconds.
strString String. The value to assign to nParameter.

Return Value bOk is TRUE if the function succeeds and FALSE if it fails.

See Also SqlGetParameter SqlGetParameterAll SqlSetParameterAll

Example
```
Call SqlSetParameter ( hSqlPrimary, DBP_PRESERVE, TRUE, strNull )
```

SqlSetParameterAll

Syntax ***bOk =* SqlSetParameterAll** *(hSql, nParameter, nNumber, strString, bNumber)*

Description Sets the value of a database parameter identified by a SQLP* constant value defined in SQL.H. This function uses the number (nNumber) and string (strString) parameters as appropriate depending on the data type of

the value of the parameter.

Parameters		
	hSql	Sql Handle. The handle that identifies a database
	nParamete	connection. Number. The database parameter to set. Specify one of the SQLP* constants defined in SQL.H.
	nNumber	Number. The value to assign to nParameter if it is a number.
	strString	String. The value to assign to nParameter if it is a string.
	bNumber	If TRUE, the parameter value is in nNumber. If FALSE, the parameter value is in strString.

Return Value bOk is TRUE if the function succeeds and FALSE if it fails.

See Also SqlGetParameter SqlGetParameterAll
SqlSetParameter

Example
```
Set bOk = SqlSetParameterAll ( hSql, nParameter, nNumber, strString, bNumber
     )
```

SqlSetResultSet

Syntax ***bOk =* SqlSetResultSet** *(hSql, bSet)*

Description Turns result set mode on or off.

Result set mode is on by default in TD Mobile.

If you are using an OLE DB connection to SQL Server, and you are executing a stored procedure that returns a result set, do not call this function with bSet=TRUE. SQL Server does not support scrollable result sets for stored procedures.

Parameters hSql Sql Handle. A handle that identifies a database connection. bSet

Boolean. Turns result set mode on (TRUE) or off (FALSE).

Return Value bOk is TRUE if the function succeeds and FALSE if it fails.

Example
```
Call SqlSetResultSet ( hSqlPrimary, FALSE )
```

SqlStore

Syntax ***bOk =* SqlStore** *(hSql, strName, strSqlCommand)*

Description Stores and names a SQLBase compiled SQL statement.

You do not need to call SqlPrepare before calling SqlStore. SqlStore compiles the SQL statement.

You can specify up to 128 bind variables. Use numeric bind variables in the SQL statement, not variable names. For example: "SELECT * FROM PRESIDENT WHERE LASTNAME = :1 AND AGE > :2;".

When you retrieve the stored command with SqlRetrieve, you specify the bind variable names in the INTO clause. For example, specify ':dfLastName' and ':dfAge' where dfLastName and dfAge are data fields on a form window.

Before TD Mobile performs a SQL execute or fetch operation, it compiles the bind and into variables which is looking up the symbols and generating the code that gets the values (for bind variables) or that fetches the values (for an into variable). By default, TD Mobile compiles:

- Bind variables at execute time
    - Into variables at fetch time

You can change this default behavior by calling SqlVarSetup which saves the current execution context. When you execute or fetch later, TD Mobile uses that execution context to resolve references to bind variables and

into variables. This means that you can use bind and into variables in a different context than where you call Sql* functions. You must call SqlPrepare for the Sql Handle before you call SqlVarSetup.

Use this function to write:

- Global functions that store bind and into variables in local variables
- A hierarchy of classes where a base class can prepare and fetch and a derived class can specify the into variables

This function does not affect the lifetime of the bind and into variables and does not guarantee that the variables will exist when you execute or fetch. You must ensure that the variables are still valid when you use them.

Parameters

hSql Sql Handle. A handle that identifies a database connection. strName String.

The name of the stored command.

strSqlCommand String. The SQL statement to compile and store.

Return Value

bOk is TRUE if the function succeeds and FALSE if it fails.

See Also

SqlDropStoredCmd
SqlRetrieve

Example

```
Set bOk = SqlStore (hSql, strName, strSqlCommand)
```

SQL OLE DB Functions

This is an alphabetical list of the SAL SQL OLEDB functions accompanied by detailed information about each function's purpose, its parameters and return value, and an example.

Function descriptions include:

- Syntax
- Description
- Parameters
- Return value
- See also
- Example

SqlCommitSession

Syntax

bOk = SqlCommitSession (hSession)

Description

This call commits the current transaction associated with the specified session. The SQL operations currently active on all the statements belonging to this session get committed.

Instead of taking a SqlHandle as its input (as done in case of the old SqlCommit function), this function takes the session handle.

This function returns TRUE if the transaction was committed successfully. If the call failed, it returns FALSE.

Parameters

hSession Session Handle. The session handle used to commit the transaction.

Return Value

bOk is TRUE if the function succeeds and FALSE if it fails.

Example

```
Actions
    ! ! Commit the transaction...
    Set bOk = SqlCommitSession (hSession) If bOk
      Call SalMessageBox ( ' Committed!!', 'Good', MB_Ok)
    Else
      Call SalMessageBox ( ' Commit failed ', 'Bad', MB_Ok)
```

See Also

SqlCreateSession SqlCreateStatement
SqlFreeSession SqlGetSessionHandle

SqlCreateSession

Syntax

***bOk* = SqlCreateSession** *(hSession, strSessionProperties)*

Description

This function creates a new session. This function takes as its input argument a string which specifies all the properties for this session. . This call returns a valid session handle if the call was successful.

All statements created using a single session belong to the same transaction. Thus, a commit call on a given session handle commits SQL operations on all the statements belonging to that session. Similarly, a rollback on a session rolls back all SQL operations on all statements belonging to that session. Instead of taking a SqlHandle as its input (as compared to the old SqlConnect function), this function takes the Session Handle.

When connected to SQLBase, or when using an OLE DB connection, a call to SqlCreateSession will create a new database connection. When connected to other databases, Gupta first checks the combination of database name, user ID, and password. For each new combination, a new database connection is created. However, if the combination has already been used in the application, only a new cursor is created, not a new database connection.

About session properties: OLE DB specifications allow a program to set specific session properties at the time of establishing a session. Using SAL, you do not have to make a separate function call to set these prooperties. TD Mobile internally sets all the properties to the values specified in this call. Session Properties also remove any dependency on the configuration (SQL.INI) file; the call to create the session itself provides all the information necessary to identify the OLE DB provider to be targeted for this session. For example, a TD Mobile application wishing to connect to SQLBase using the TD Mobile OLE DB Data Provider specifies, at the minimum, the following session property:

```
Set strSessionProp = "Provider=SQLBASEOLEDB;"
```

A more explicit version of the same example might be:

```
Set strSessionProp = "Provider=SQLBASEOLEDB;Data Source=Island;User
IDr=sysadm;Password=sysadm;"
```

Connection string information may be overridden by system variables

In all cases described above, any missing information in the final connection string is obtained from the system variables SqlDatabase, SqlUser, and SqlPassword. In addition, even if the connection string does contain information about user, password, and database, that information may be overridden. If there are any values in variables SqlDatabase, SqlUser, and SqlPassword, those values will be used in preference to anything that is already present in the connection string, regardless of whether that connection string came from variable SqlConnectOptions, from variable SqlUDL, from a UDL file, or from an actual string passed as a parameter to this function.

Parameters

hSession	Session Handle. The session handle created as a result of this call.
strSessionProperties	String. The string that specifies the session properties for this session. There are several possible behaviors for this parameter. If it is null, this function looks at system variable SqlConnectOptions which is meant to hold connection information, then in SqlUDL to get connection information. SqlUDL may contain the name of a UDL file or the name of an OLE DB provider. If SqlUDL is also null, then this session makes a connection using the SQL API and routers, not OLE DB. If strSessionProperties is not null, it may be a file name ending in .UDL - in this case, the function reads that file to obtain connection information. Otherwise, when strSessionProperties is not null, it is presumed that this parameter is a string that contains connection information. In all the cases above, if the resulting connection string is missing the database name, user ID or password, then values are obtained from SAL global variables SqlDatabase, SqlUser and SqlPassword respectively.

Note: SqlConnectOptions supports the following name value pairs: PROVIDER
INPUTMSGSIZE
OUTPUTMSGSIZE
AUTOCOMMIT TXNISOLATION

Return Value

bOk is TRUE if a new session was created successfully. If the call failed, it returns FALSE.

Example

```
If dfSessionProp != ''
 Set strSessionProp = strSessionProp || dfSessionProp
 Set SqlDatabase = dfDatasource
 Set SqlUser = dfUser
 Set SqlPassword = dfPassword
 Set bOk = SqlCreateSession (hSession, strSessionProp)
```

See Also SqlCommitSession SqlCreateStatement SqlFreeSession SqlGetSessionHandle

SqlCreateStatement

Syntax

***bOk* = SqlCreateStatement** *(hSession, hSql)*

Description

This call creates a new statement belonging to the specified session. The Sql Handle parameter specified here is the same as what the current SqlConnect call returns. There can be any number of statements within a session; there is no limit on this number.

This call returns a statement handle if the call was successful. To free a statement, the existing SqlDisconnect call needs to be used.

Parameters

hSession	Session Handle. The Session handle used to create the statement.
hSql	Sql Handle The Sql handle used to associate any number of statements to a session.

Return Value

bOk is TRUE if a statement was created successfully. If the call failed, it returns FALSE.

Actions

```
Set bOk = SqlCreateStatement (hSession, hSql)
```

See Also SqlCommitSession SqlCreateSession SqlFreeSession SqlGetSessionHandle

SqlFreeSession

Syntax ***bOk =* SqlFreeSession** *(hSession)*

Description This call frees the session. If there are any open statements belonging to this session, they are closed before the session is freed.

Parameters

hSession	Session Handle. The session handle used to commit the transaction.

Return Value bOk is TRUE if the specified session was freed successfully. If the call failed, it returns FALSE.

Example

```
Actions
   If (hSession)
     Call SqlFreeSession (hSession)
```

See Also SqlCommitSession SqlCreateSession SqlCreateStatement SqlGetSessionHandle

SqlGetCmdOrRowsetPtr

Syntax ***bOk =* SqlGetCmdOrRowsetPtr** *(hSql, bCmdOrRowset, numOLEDBPtr)*

Description This function gives the caller either the ICommand or the IRowset interface pointer of the Command or the Rowset OLE DB object.

Once you get the interface pointer, you can pass it to an external DLL and use it as needed (for example, to access interfaces/methods that we do not expose in SAL).

Parameters

hSql	Sql Handle. The Sql handle associated with the Command or the Rowset object.
bCmdOrRowset	BOOLEAN. If set to TRUE, this function gives the user the ICommand interface pointer. If set to FALSE, this function gives the user the IRowset interface pointer.
numOLEDBPtr	Number. This contains the interface pointer as specified if the function was successful. It contains NULL if there is no rowset associated with this Sql handle yet, and if the user asks for the IRowset interface pointer.

Return Value bOk is TRUE if the function succeeds and FALSE if it fails.

Example

```
Actions
    ! ! Get the Rowset object ptr.  The code returns the
          IRowset interface ptr
    Set bOk =  SqlGetCmdOrRowsetPtr( hSql, 0, dfRSPtr )
```

See Also SqlGetDSOrSessionPtr

SqlGetDSOrSessionPtr

Syntax ***bOk =* SqlGetDSOrSessionPtr** *(hSql, bDSOrSession, numOLEDBPtr)*

Description This function gives the caller the IDBInitialize interface pointer of the Data Source OLE DB object or the IDBCreateSession interface pointer of the Session OLE DB object.

Once you get the interface pointer, you can pass it to an external DLL and use it as needed (for example, to

access interfaces/methods that we do not expose in SAL).

Parameters

hSql — Sql Handle. The Sql handle associated with the interface pointer.

bDSOrSession — BOOLEAN. If set to TRUE, this function gives the user the IDBInitialize interface pointer. If set to FALSE, this function gives the user the IDBCreateSession interface pointer.

numOLEDBPtr — Number. NULL if there is no rowset associated with this Sql Handle yet, and if the user asks for the IDBInitialize interface pointer.

Return Value

bOk is TRUE if the function succeeds and FALSE if it fails.

Example

```
Actions
            !!Get the DataSource object ptr.  The code returns the
              IDBInitialize interface ptr
 Set bOk =  SqlGetDSOrSessionPtr( hSession, 1, dfDSPtr )
```

See Also

SqlGetCmdOrRowsetPtr

SqlGetNextSPResultSet

Syntax

bOk = SqlGetNextSPResultSet (hSql, strIntoList, bEndOfRS)

Description

If a stored procedure invoked by calling SqlPrepareSP (and later executed by calling SqlExecute or SalTblPopulate) returns more than one result set, the application should call this function to get the second and subsequent result sets.

Separate the variables listed in strIntoList with commas and precede each variable name with a colon. If the stored procedure returns zero rows, the variables in strIntoList keep whatever values they had before the call to this function.

If a result set is returned its associated cursor points to just before the first row of the result set. To set the INTO variables in strIntoList to the column values of the first row, call SqlFetchNext. To obtain subsequent rows in the result set, repeatedly call SqlFetchNext.

Once all the rows in a given result set have been retrieved, get the next result set (if any) by again calling SqlGetNextSPResultSet.

Parameters

hSql — Sql Handle. The sql handle on which the Stored Procedure should be executed.

strIntoList — String. String variable that contains the Into variables for any result set generated by the Stored Procedure. If the caller knows that the stored procedure does not generate any result set, this can be set to strNull. If passed, this should be a comma-separated list of variables and precede each variable name with a colon.

bEndOfRS — BOOLEAN. If SqlGetNextSPResultSet is called and there are no more result sets, the function will return FALSE and bEndOfRS is set to TRUE. If there are more result sets, bEndOfRS will be set to FALSE.

Return Value

bOk is TRUE if the function succeeds and FALSE if it fails.

Example

```
Actions
  Set bOk = SqlGetNextSPResultSet( hSql, '  :dfString  ', bEORS)
```

See Also

SqlCloseAllSPResultSets
SqlPrepareSP

SqlGetSessionErrorInfo

Syntax

***bOk =* SqlGetSessionErrorInfo** *(hSession, numErrorNumber, strErrorDescription, strSqlState)*

Description This call returns the error information associated with the specified session.

Use this function if any of the newly introduced SAL functions which take the Session handle as the input return FALSE.

Parameters hSession Session Handle. The Session handle passed to the SAL call that failed.

numErrorNumber Number The error number. strErrorDescription

String. The error description. strSqlState String. The Sql state.

Return Value

bOk is TRUE if the error information was retrieved successfully. If the call failed, it returns FALSE.

Example

```
Actions
  Set SessionProperties="Provider=DataDirect.Oracle8OLEDBProvider;" Set bok =
  SqlCreateSession(strSessionProp,SessionProperties)
  If bok
    Call SqlCreateStatement (strSessionProp, hSql)
    Set bok = SqlPrepareAndExecute (hSql,'drop table test ( col1 int ) ')
    If bok
          ..
       Else
      Set nError =  SqlError( hSql )
      Call SqlGetStatementErrorInfo( strSessionProp, nError1, HStr, HStr1 ) Else
    Call SqlGetSessionErrorInfo( strSessionProp, nError1, HStr, HStr1 )
```

See Also SqlGetStatementErrorInfo

SqlGetSessionHandle

Syntax ***bOk =* SqlGetSessionHandle** *(hSql, hSession)*

Description This call returns the session handle to which the specified statement handle belongs.
The SqlHandle must have been created using SqlCreateStatement function and not by calling SqlConnect function.

Parameters hSql Sql Handle. The Sql handle used to associate any number of statements to a session.

hSession Session Handle. The Session handle of the statement. Return Value bOk is TRUE if the function was successful. If the call failed, it returns FALSE. Example

```
   Actions
! ! Create the statement
Set bOk = SqlCreateStatement (hSession, hSql)
```

See Also SqlCommitSession SqlCreateSession
SqlCreateStatement SqlFreeSession

SqlGetSessionParameter

Syntax ***bOk =* SqlGetSessionParameter** *(hSession, numPropertyID, numValue, strValue)*

Description This function gets the value of the specified session property.

It takes as input the session handle and the property ID. This function will know the data type of the specified property ID and will accordingly return either the number value or the string value.

Parameters hSession Session Handle. The session handle. numPropertyID Number. The number value of the property ID. numValue Number. The number value of the

property ID. strValue String. The string value of the property ID.

Return Value bOk is TRUE if successful. It will return FALSE if it failed.

Example

```
Actions
            Set bOk = SqlGetSessionParameter( hSession, dfPropID,
            dfIntValue, dfStrValue)
```

See Also SqlSetSessionParameter

SqlGetStatementErrorInfo

Syntax ***bOk =* SqlGetStatementErrorInfo** *(hSql, numErrorNumber, strErrorDescription, strSqlState)*

Description This call returns the error information associated with the specified statement handle (command/cursor).

Note that this function will work with Sql Handles created either with SqlCreateStatement or SqlConnect. In the case of Sql Handles created with SqlConnect, the SQLState will be always NULL.

Parameters hSql Sql Handle. The Sql handle passed to the SAL call that failed. numErrorNumber

Number The error number.

strErrorDescription String. The error description. strSqlState

String. The Sql state.

Return Value bOk is TRUE if the error information was retrieved successfully. If the call failed, it returns FALSE.

Example

```
Actions
    Set SessionProperties="Provider=DataDirect.Oracle8OLEDBProvider;" Set bok =
    SqlCreateSession(strSessionProp,SessionProperties)
    If bok
      Call SqlCreateStatement (strSessionProp, hSql)
      Set bok = SqlPrepareAndExecute (hSql,'drop table test ( col1 int ) ')
      If bok
            …
     Else
        Set nError =  SqlError( hSql )
        Call SqlGetStatementErrorInfo( strSessionProp, nError1, HStr, HStr1 ) Else
      Call SqlGetSessionErrorInfo( strSessionProp, nError1, HStr, HStr1 )
```

See Also SqlGetSessionErrorInfo

SqlPrepareSP

Syntax ***bOk =* SqlPrepareSP** *(hSql, strStoredProc, strIntoList)*

Description This SAL function prepares a stored procedure invocation statement. It handles any input parameters passed to it by the caller.

The function also handles output parameters, but the output parameters will not be updated after a successful execution. The values of the SAL variables specified for any output parameter are updated only after any result set generated by the stored procedure has been completely processed.

If you are calling a Microsoft SQL Server stored procedure, be sure that front-end result sets have been disabled first.

Parameters hSql Sql Handle. The sql handle on which the Stored Procedure should be executed.

	strStoredProc	String. String variable which contains the stored procedure name and any optional input or output parameters. This string can either be in the ODBC calling syntax () or in the native database format. For ODBC syntax with SQL Server procedures that have no input parameters and no output parameters, you must drop the parentheses following the procedure name. A set of empty parentheses will cause a SQL error in this case. For example:
	strIntoList	String String variable that contains the Into variables for any result set generated by the Stored Procedure. If the caller knows that the stored procedure does not generate any result set, this can be set to strNull. If passed, this should be a comma-separated list of variables and precede each variable name with a colon.

Return Value bOk is TRUE if the function succeeds and FALSE if the prepare fails. Once the stored procedure has been prepared, it can be executed either by calling SqlExecute on the same Sql Handle or by calling SalTblPopulate.

Example

```
Actions
 ! ! Now time to prepare the statement...
 Set bOk = SqlPrepareSP  (hSql,  dfSQL, STRING_Null ) If bOk
   Set numInput = dfInteger
   Set bOk = SqlExecute ( hSql )
```

See Also SqlCloseAllSPResultSets
SqlGetNextSPResultSet

SqlRollbackSession

Syntax ***bOk =* SqlRollbackSession** *(hSession)*

Description This call rolls back the current transaction associated with the specified session.

The SQL operations currently active on all the statements belonging to this session get rolled back.

Parameters hSession Session Handle. The session handle used to commit the transaction.

Return Value bOk is TRUE if the transaction was rolled back successfully. If the call failed, it returns FALSE.

SqlSetSessionParameter

Syntax ***bOk =* SqlSetSessionParameter** *(hSession, numPropertyID, numValue, strValue)*

Description This function sets the value of the specified session property.

It takes as input the session handle and the property ID. This function will know the data type of the specified property ID and will accordingly use either the number value or the string value.

Parameters hSession Session Handle. The Session handle. numPropertyID

Number. The property ID.

numValue Number. The number value of the property ID. strValue String.

The string value of the property ID.

Return Value bOk is TRUE if successful. It will return FALSE if it failed.

Example

```
Actions
```

```
Actions
 Set bOk = SqlSetSessionParameter( hSession, dfPropID, dfIntValue, dfStrValue)
```

See Also SqlGetSessionParameter

SQL Oracle PL/SQL Functions

This is an alphabetical list of the SAL functions which support anonymous PL/SQL blocks accompanied by detailed information about each function's purpose, its parameters and return value, and an example.

Function descriptions include:

- Syntax
- Description
- Parameters
- Return value
- See also
- Example

SqlOraPLSQLPrepare

Syntax *bOk* **= SqlOraPLSQLPrepare** *(hSql, strAnonymousPLSQLBlock)*

Description This function compiles the anonymous PL/SQL block. This function looks very much like the regular SqlPrepare function. But the underlying code is meant specifically for handling Oracle PL/SQL blocks.

Parameters

hSql SqlHandle. A handle that identifies a database connection.

strAnonymousPLSQLBlock String. The actual anonymous PL/SQL block that the user wants to prepare. That will also contain the input and output variables.

Return Value bOk is TRUE if the function succeeds and FALSE if it fails.

Example

```
Set strAnonymousPLSQLBlock = ' BEGIN
  Pkg1.Proc1 ( :nInput1, :sOutput1, :nOutput2 ); Pkg1.Proc2 (
  :nOutput2 );
  Pkg1.Proc3 ( :sOutput1 );
  END; '
bOk = SqlOraPLSQLPrepare ( hSql, strAnonymousPLSQLBlock)
```

Note: If this call is made to a non-Oracle connection, the function returns FALSE. This call needs a newer version of the Oracle router. If the router being used is not capable of supporting this call, an error "This call needs a newer version of Oracle router." is returned.

SqlOraPLSQLExecute

Syntax ***bOk* = SqlOraPLSQLExecute** *(hSql)*

Description This function executes the anonymous PL/SQL block that was prepared using SqlOraPLSQLPrepare. If the execution succeeds, then all output parameters from the PL/SQL block are updated by the time the control returns to the user.

Parameters

hSql Sql Handle. The sql handle associated with the prepared Anonymous PL/SQL block.

Return Value bOk is TRUE if the function succeeds and FALSE if it fails.

Example

```
Set strCMD = ' BEGIN
  Pkg1.Proc1 ( :nInput1, :sOutput1); Pkg1.Proc2 (
  :sOutput1 ); Pkg2.Proc1 ( :nInput1 );
  END; '
Set bOK = SqlOraPLSQLPrepare (hSql, strCMD) Set nInput1 = 100
Set bOk = SqlOraPLSQLExecute ( hSql )
```

SqlOraPLSQLStringBindType

Syntax

***bOk* = SqlOraPLSQLStringBindType** *(hSql, strBindName, nBindType)*

Description

This function informs the Oracle router the specific type of the STRING array bind variable that is being used in the prepared anonymous PL/SQL block. By default, any STRING array is bound to Oracle as a VARCHAR type. If the PL/SQL table is of type CHAR, this function needs to be called. If the PL/SQL table is either VARCHAR or STRING, there is no need to call this function. Call this function after doing a SqlOraPLSQLPrepare but before doing the SqlOraPLSQLExecute. Since the binding is done for every execute, you need to call this function every time you execute.

Parameters

hSql	Sql Handle. The sql handle associated with the prepared Anonymous PL/SQL block.
strBindName	String. The name of the bind variable (the input or the output parameter) as specifed in the sql statement passed for the SqlOraPLSQLPrepare command.
nBindType	Number. The PL/SQL table datatype. Specify 5 if the table is either VARCHAR or STRING. Specify 97 if the PL/SQL table is of type CHAR.

Return Value

bOk is TRUE if the function succeeds and FALSE if it fails.

Example

```
! Data type declarations
!
String: strCMD Number: nInput1
String: sOutput1[1:100]
Boolean: bOk
!
! Code follows
!
Set strCMD = ' ArrayPkg.Proc1 (:nInput1, :sOutput1); ' Set bOk =
SqlOraPLSQLPrepare (hSql, strCMD)
Set nInput1 = 100
Set bOk = SqlOraPLSQLStringBindType (hSql, 'sOutput1', 97) Set bOk =
SqlOraPLSQLExecute (hSql)
If bOk
  ! now the output parameter is availableÖ Call SalMessageBox
  ( sOutput1[1],
    'Output parameter #1 from PL/SQL', MB_Ok)
```

Note: This function needs to be called only for string array parameters. If the PL/SQL block uses only scalars, irrespective of whether is CHAR or VARCHAR or STRING, this function is not needed. If the parameter is an array of type CHAR this function needs to be called. If this is not called and SqlOraPLSQLExecute is called, Oracle server will return errors ORA-6550 and PLS-00418 - "Array bind type must match PL/SQL table row type".

String Functions

This is an alphabetical list of the SAL string functions accompanied by detailed information about each function's purpose, its parameters and return value, and an example.

Function descriptions include:

- Syntax
- Description
- Parameters
- Return value
- See also
- Example

SalStrCompress

Syntax ***bOk =* SalStrCompress** *(strString)*

Description Compresses the specified string. Use this function to compress strings for storage on disk or in the database. Use this function for long strings, or when storing images and so on.

Note: when the last character of the string is a null character, you may encounter an error if the compressed string is inserted into a database table, fetched back from that table, and used in SalStrUncompress. This is because some databases will not store the terminating null character. Thus the database string is now shorter by one

character, which conflicts with the original string length. To prevent this error, check for the null character and, if present, add code like this line after the call to SalStrCompress:

```
Call SalStrSetBufferLength( sBuffer, SalStrGetBufferLength(
  sBuffer ) + 1 )
```

Parameters strString Receive String. The string to compress.

Return Value bOk is TRUE if the function succeeds and FALSE if it fails.

See Also SalStrUncompress

Example

```
Set bOk = SalStrCompress ( strString )
```

SalStrFirstC

Syntax ***bOk =* SalStrFirstC** *(strString, nChar)*

Description Extracts the first character of a string and initializes a numeric parameter with its value. If the application has DBCS (double-byte character set) enabled, the number represents the integer value of the 16-bit character. Otherwise, the number represents the ASCII value of the 8-bit character.

You must use this function in place of SalStrLop if the input string contains DBCS or 16-bit characters. If the character returned is a 16-bit character, the leading byte of the character is in the high-order byte, and the trailing byte of the character is in the low- order byte.

Use SalNumberHigh to get the leading byte and SalNumberLow to get the trailing byte.

Parameters strString Receive String. The string whose first character is lopped off. nChar

Receive Number. The first character of strString.

Return Value bOk is TRUE unless strString is empty or invalid.

See Also
SalNumberHigh SalNumberLow
SalNumberToChar SalStrLop

Example

```
Actions
    Set dfString = 'ABC'
    Call SalStrFirstC ( dfString, dfNumChar )
    ! Now dfString = 'BC' and dfNumChar = 65
```

SalStrGetBufferLength

Note: This API is deprecated due TD Mobile's switch from ANSI to Unicode. See SalGetBufferLength in the in-build help.

Syntax
***nLength* = SalStrGetBufferLength** *(strString)*

Description
Returns the current buffer length of a string.

TD Mobile stores string variables in buffers. The length includes the string's null termination character.

Parameters
strString String. The string whose buffer length you want.

Return Value
nLength is the length of strString's buffer.

See Also
SalStrLength
SalStrSetBufferLength

Example

```
Set nBufferLength = SalStrGetBufferLength
                (strNumbers )
```

SalStrIsValidCurrency

Syntax
***bOk* = SalStrIsValidCurrency** *(strMoney, nPrecision, nScale)*

Description
Verifies that an entire character string represents a valid currency value.
TD Mobile validates the string based on the current settings for the keywords sDecimal, sThousands, sCurrency, iNegCurr, and iCurrency in the [INTL] section of WIN.INI.

Parameters
strMoney String. A string that contains a currency value. nPrecision

Number. The number of digits to display.

nScale Number. The number of digits to the right of the decimal point.

Return Value
bOk is TRUE if strMoney is a valid currency value and FALSE otherwise.

See Also
SalStrIsValidDateTime
SalStrIsValidNumber

Example

```
Set bOk1 = SalStrIsValidCurrency ( '$120.00' )
```

SalStrIsValidDateTime

Syntax
***bOk* = SalStrIsValidDateTime** *(strDateTime)*

Description
Verifies that an entire character string represents a valid date/time value.
TD Mobile validates the string based on the current settings for the keywords sShortDate, sLongDate, s1159, s2359, and sTime in the [INTL] section of WIN.INI.

Parameters
strDateTime String. A string that contains a date/time value.

Return Value
bOk is TRUE if strDateTime is a valid date/time value and FALSE otherwise.

See Also SalStrIsValidCurrency
SalStrIsValidNumber

Example
```
Set bOk2 = SalStrIsValidDateTime ( '2/2/91' )
```

SalStrIsValidNumber

Syntax ***bOk* = SalStrIsValidNumber** *(strNumber)*

Description Verifies that an entire character string represents a valid number value. TD Mobile validates the string based on the current settings for the keywords sDecimal and sThousands in the [INTL] section of WIN.INI.

Parameters strNumber String. A string that contains a number value. Return Value bOk is TRUE if strNumber is a valid number value and FALSE otherwise. See Also SalStrIsValidCurrency
SalStrIsValidDateTime

Example
```
Set bOk1 = SalStrIsValidNumber ( '120.00' )
```

SalStrLeft

Syntax ***nLength* = SalStrLeft** *(strSource, nExtract, strTarget)*

Description Returns a substring of a specified length starting at position zero (0), the left-most character in the string.

Parameters strSource String. The string from which to extract characters.

nExtract Number. The number of characters to extract from strSource. strTarget

Receive String. The substring.

You can specify the same string for both strSource and strTarget.

Return Value nLength is the length of the new string. strTarget is the substring.

See Also SalStrMid
SalStrMidX
SalStrRight
SalStrLeftX

Examples
```
Set nLength = SalStrLeft ( 'LEFT01234', 4, strTarget )
```

SalStrLeftX

Syntax ***strTarget* = SalStrLeftX** *(strSource, nExtract)*

Description Returns a substring of a specified length starting at position zero (0), the left-most character in the string.

Parameters strSource String. The string from which to extract characters.

nExtract Number. The number of characters to extract from strSource. strTarget

Receive String. The substring.

You can specify the same string for both strSource and strTarget.

Return Value nLength is the length of the new string. strTarget is the substring.

See Also SalStrMid SalStrMidX SalStrRight
SalStrLeft

Examples
```
Set strTarget = SalStrLeftX ( 'LEFT01234', 4)
```

SalStrLength

Syntax ***nLength* = SalStrLength** *(strString)*

Description Returns a string's length.

Strings are stored internally in TD Mobile with a null termination character. The null terminator is not included in the length.

Parameters strString String. The string whose length you want.

Return Value nLength is the length of strString.

See Also SalStrGetBufferLength

Example
```
Actions
    Set strNumbers = '1234567890'
    Set nStringLength = SalStrLength ( strNumbers )
```

SalStrLop

Syntax ***nCharacter* = SalStrLop** *(strString)*

Description Returns the ASCII numeric value of the first character of a string in decimal format.
This function removes the first character of the string.

Parameters strString Receive String. The input string without the first character.

Return Value nCharacter is the ASCII value of the first character of strString. When strString is null, nCharacter is equal to zero (0).

See Also SalNumberToChar
SalStrFirstC

Example
```
Actions
    Set strString = 'ABC'
    Set nCharacter = SalStrLop ( strString )
```

SalStrLower

Syntax ***nLength* = SalStrLower** *(strSource, strTarget)*

or

***strTarget* = SalStrLowerX** *(strSource)*

Description Converts a string to lowercase.

Parameters strSource String. The string to convert . strTarget Receive

String. The lowercase string.

You can specify the same string for both strSource and strTarget.

Return Value nLength is the length of strTarget. strTarget is the lowercase

string.

See Also SalStrUpper
SalStrUpperX

Examples
```
Actions
    Set nLength = SalStrLower ( 'LOWERCASE', strTarget )
or
Actions
    Set strTarget = SalStrLowerX ( 'LOWERCASE' )
```

SalStrMid

Syntax ***nLength* = SalStrMid** *(strSource, nStartPos, nLength, strTarget)*

or

***strTarget* = SalStrMidX** *(strSource, nStartPos, nLength)*

Description Returns a substring, starting at a specified position and containing a specified number of characters.

Parameters

strSource	String. The source string.
nStartPos	Number. The starting position of the substring (zero is the first position) in strSource.
nLength	Number. The number of characters to put in the substring. strTarget

Receive String. The substring.

You can specify the same string for both strSource and strTarget.

Return Value nLength is the length of the substring. strTarget is the substring.

See Also SalStrLeft SalStrLeftX SalStrRight

Examples

```
Set nLength = SalStrMid ('012ABC345', 3, 3, strTarget) Or
Set strTarget = SalStrMidX ( '012ABC345', 3, 3 )
! strTarget = 'ABC'
```

SalStrProper

Syntax ***nLength* = SalStrProper** *(strSource, strTarget)*

or

***strTarget* = SalStrProperX** *(strSource)*

Description Converts a string to a proper name. In a proper name, the first letter of each word is uppercase; the remaining letters are lowercase.

Parameters

strSource	String. The string to convert.
strTarget	Receive String. The converted string.

You can specify the same string for both strSource and strTarget.

Return Value nLength is the length of strTarget. strTarget is the converted

string.

Examples

```
Set nLength = SalStrProper ( 'JOHN L. SMITH', strTarget )
or
Set strTarget = SalStrProperX ( 'JOHN L. SMITH')
```

SalStrRepeat and SalStrRepeatX

Syntax ***nLength* = SalStrRepeat** *(strSource, nTimes, strTarget)*

or

***strTarget* = SalStrRepeatX** *(strSource, nTimes)*

Description Concatenates a string with itself a specified number of times.

Parameters

strSource	String. The source string.
nTimes	Number. The number of times to concatenate strSource with itself.
strTarget	Receive String. The new string.

You can specify the same string for both strSource and strTarget.

Return Value nLength is the length of strTarget. strTarget is the new string.

Examples

```
Actions
   Set nLength = SalStrRepeat ( 'ABC*', 3, strTarget )

or

Actions
             Set strTarget = SalStrRepeatX '( 'ABC*', 3)
```

SalStrReplace and SalStrReplaceX

Syntax ***nReturn* = SalStrReplace** *(strSource, nStartPos, nLength, strReplace, strTarget)*

or

***strTarget* = SalStrReplaceX** *(strSource, nStartPos, nLength, strReplace)*

Description Replaces characters in one string with characters from another string.

Parameters

strSource	String. The source string that contains characters to replace.
nStartPos	Number. The position in strSource at which to begin replacing characters.
nLength	Number. The number of characters to replace. strReplace

String. The replacement string.

strTarget	Receive String. The new string.

Return Value nReturn is the length of strTarget. strTarget is the new string.

Examples

```
Message Actions
              Actions
            Set nReturn = SalStrReplace ('far', 0, 1, 'be', strTarget )
            ! strTarget = 'bear' and nReturn = 4

              Actions
            Set strTarget = SalStrReplaceX ('bear', 0, 2, 'f' )
            !strTarget = 'far'
```

SalStrRight and SalStrRightX

Syntax ***nLength* = SalStrRight** *(strSource, nLength, strTarget)*

or

***strTarget* = SalStrRightX** *(strSource, nLength)*

Description Returns a string of specified length, starting with the last character in the string.

Parameters

strSource	String. The source string.
nLength	Number. The number of characters to extract. strTarget

Receive String. The new string.

You can specify the same string for both strSource and strTarget.

Return Value nLength is the length of strTarget. strTarget is the new string.

See Also SalStrLeft SalStrLeftX SalStrMid
SalStrMidX

Example
```
Actions
    Set nLength = SalStrRight ( '123RIGHT', 5, strTarget )

or

Actions
    Set StrTarget = SalStrRightX ( '123RIGHT', 5 )
```

SalStrScan

Syntax ***nOffset* = SalStrScan** *(strString1, strString2)*

Description Searches for and returns the offset of a specified substring. If there is more than one instance of the string being searched for, only the offset of the first instance is returned.

Parameters

strString1	String. The string to search. The first character in the string is at offset zero (0).
strString2	String. The string to search for. Case is disregarded in the search. You can use pattern matching characters. The percent character (%) matches any set of characters. The underscore character (_) matches any single character. The use of a backslash(\) with SalStrScan differs when searching for a backslash, percent, or underscore character. Its usage also differs depending on whether or not the second parameter is a string literal. When searching for a backslash and strString2 is a string literal, you need four backslashes: SalStrScan ('This is a \\', '\\\\') When searching for a percent character or an underscore character and strString2 is a string literal, you need two backslashes: SalStrScan ('This is a %', '\\%') SalStrScan ('This is an _', '_') Even if strString2 is not a string literal, you need a single backslash to search for a percent character or an

Return Value nOffset is a number that indicates the offset (0 origin) of strString2 in strString1. If TD Mobile does not find strString2 in strString1, SalStrScan returns a -1.

Example
```
Set nOffset = SalStrScan ( '012AbC345', 'ABC' )
```

SalStrSetBufferLength

Note: This API is deprecated due TD Mobile's switch from ANSI to Unicode. See SalSetBufferLength in the in-build help.

Syntax ***bOk* = SalStrSetBufferLength** *(strString, nLength)*

Description Sets the buffer string length to the parameter value and allocates memory. If strString is expected to contain a string value, rather than binary bytes, be sure to set nLength equal to 1 plus the expected number of

characters, to accommodate the null terminator. This is only needed if you want to pass a Receive String to an external function.

Note that after calling this function, if you subsequently assign a value to the string using an ordinary operation like `Set sExample = 'some text'`, the buffer length of the string will change to match the number of characters assigned, plus one for the null terminator. If you then call an external function that was expecting the original buffer length, you risk the chance of memory corruption through writing text beyond the buffer length.

If a string already has characters assigned to it before you call SalStrSetBufferLength, and you then call the function using a length that is less than the present buffer length, you will truncate the string and lose the null terminator character. This may cause problems when you pass the string to an external function.

Parameters

strString Receive String. The string whose buffer length you want to set. nLength

Number. The length of strString.

Return Value

bOk is TRUE if the function succeeds and FALSE if it fails.

See Also

SalSetMaxDataLength

SalStrGetBufferLength

Example

```
Set bOk = SalStrSetBufferLength( recvString, 10 ) ! 9 chars
```

SalStrToDate

Syntax ***dtDateTime =* SalStrToDate** *(strString)*

Description Converts a string to a date/time value.This function uses the system date format to convert a date string. If you want to be format independent, use SalDateConstruct.

Parameters strString String. The string to convert.

Return Value dtDateTime is the date/time value converted from strString.

See Also SalDateToStr

Example
```
Set dtDateTime = SalStrToDate ( strDateTime )
```

SalStrTokenize

Syntax ***nNumTokens =* SalStrTokenize** *(strSource, strStartDel, strEndDel, strTokenArray)*

Description Parses a string into substrings (tokens) based on specified start and end delimiters.
TD Mobile uses delimiters to recognize the beginning and end of each substring.

TD Mobile interprets the first non-start delimiter character as the beginning of a substring, and skips any start delimiters that precede this character. For example, if '!' is a start delimiter, the strings 'Hello' and '!!!Hello' produce the same token: 'Hello'.

If the first non-start delimiter character is an end delimiter character, TD Mobile interprets it as a null substring. This is useful for comma-separated data where ',' is an end delimiter. TD Mobile recognizes that the records 'data1,data2,,data4' and
',data2,data3,data4' have four tokens each, one of which is null.

Once TD Mobile finds the beginning of a substring, it interprets all characters that follow as elements of the substring until it finds an end delimiter. For example, if '!' is a start delimiter and '?' is an end delimiter, the string 'abc!def?ghi!' produces the tokens: 'abc!def' and 'ghi!'. Although the exclamation point is a start delimiter, TD Mobile correctly interprets them as elements of the substring.

Parameters strSource String. The string to parse.

strStartDel String. A string that contains the start delimiter characters.

Pass an empty string ('') to specify the lack of a start delimiter. strEndDel

String. A string that contains the end delimiter characters.

Pass an empty string ('') to specify the lack of an end delimiter.

strTokenArray String Array. The handle (or name) of an array of substrings created from strSource.

Return Value nNumTokens is the number of substrings created. nNumTokens is zero (0) if no substrings are created, or if an error occurs.

Example
```
Set dfNumTokens = SalStrTokenize( dfSource1, '', ',', astrToken1 )
```

SalStrToMultiByte

Syntax ***bOk =* SalStrToMultiByte** *(strInput,strOutput,nEncoding)*

Description Converts a unicode string to a multibyte string.

Parameters strInput String. The string to convert.

strOutput String. The output string.

nEncoding Number. The encoding to use. The following pre-defined number constants have been defined:

ENC_ANSI
ENC_MACCP
ENC_OEMCP
ENC-UTF7
ENC_UTF8

Return Value bOk is TRUE if the function succeeds and FALSE if it fails.

See Also SalStrToWideChar

SalStrToNumber

Syntax ***nNumber* = SalStrToNumber** *(strString)*

Description Converts a string to a number.

Parameters strString String. The string to convert. Return Value nNumber is the number resulting from the conversion. See Also SalNumberToStr

Example
```
Set nNumber = SalStrToNumber ( '100.22' )
```

SalStrToWideChar

Syntax ***bOk* = SalStrToWideChar** *(strInput,strOutput,nEncoding)*

Description Converts an multibyte string to a unicode string.

Parameters strInput String. The string to convert. strOutput String. The output string.
nEncoding Number. The encoding to use. The following pre-defined number constants have been defined:
ENC_ANSI ENC_MACCP ENC_OEMCP
ENC_UTF7
ENC_UTF8

Return Value bOk is TRUE if the function succeeds and FALSE if it fails.

See Also SalStrToMultiByte

SalStrTrim

Syntax ***nNewLength* = SalStrTrim** *(strSource, strTarget)*

or

***strTarget* = SalStrTrimX** *(strSource)*

Description Strips leading and trailing blanks and compresses multiple spaces and tabs within a string to single spaces.

Parameters strSource String. The original string. strTarget Receive String. The new string.

You can specify the same string for both strSource and strTarget.

Return Value nNewLength is the length of strTarget. strTarget is the new string.

Example
```
Set nLength = SalStrTrim ( ' 1 2  3 ', strTarget )
```

SalStrTrimX

Syntax ***strTarget* = SalStrTrimX** *(strSource)*

Description Strips leading and trailing blanks and compresses multiple spaces and tabs within a string to single spaces.

Parameters strSource String. The original string. strTarget Receive String. The new string.

You can specify the same string for both strSource and strTarget.

Return Value nNewLength is the length of strTarget.
strTarget is the new string.

Example
```
Actions
   Set nLength = SalStrTrim ( ' 1 2  3 ', strTarget )
```

SalStrUncompress

Syntax ***bOk =*** **SalStrUncompress** *(strString)*

Description Decompresses the specified string. Use this function to decompress a string that you compressed with SalStrCompress.

Parameters strString Receive String. The string to decompress.

Return Value bOk is TRUE if the function succeeds and FALSE if it fails.

See Also SalStrCompress

Example
```
Set bOk = SalStrUncompress ( strString )
```

SalStrUpper

Syntax ***nLength =*** **SalStrUpper** *(strSource, str Target)*

Description Converts a string to uppercase.

Parameters strSource String. The string to convert.

strTarget Receive String. The uppercase string.

You can specify the same string for both strSource and strTarget.

Return Value nLength is the length of strTarget. strTarget is the uppercase string.

See Also SalStrLower

Example
```
Set nLength = SalStrUpper ( 'uppercase', strTarget )
```

SalStrUpperX

Syntax ***strTarget =*** **SalStrUpperX** *(strSource)*

Description Converts a string to uppercase.

Parameters strSource String. The string to convert.

You can specify the same string for both strSource and strTarget.

Return Value This function returns the uppercase string in strTarget.

See Also SalStrLower

Example
```
Actions
      Set strTarget = SalStrUpperX ( 'uppercase' )
```