Wonderware® FactorySuite™

SQL Access Manager User's Guide

Revision A

Last Revision: August 2002

Invensys Systems, Inc.

Invensys Systems, Inc.
33 Commercial Street
Foxboro, MA 02035
(949) 727-3200
http://www.wonderware.com

Contents

C H A P T E R 1

SQL Access Manager

Wonderware® FactorySuite™ SQL Access Manager allows you to access, modify, create and delete tables in a database. A database stores information in tables that share a common attribute or field. Structured Query Language (SQL) is the language used to access that information.

Contents

Introduction

The InTouch SQL Access Manager add-on program is designed to easily transfer data, such as batch recipes from a SQL database to an InTouch application. It also facilitates the transfer of run-time data, alarm status or historical data from InTouch to the SQL database. For example, after a machine cycle is completed, a company may need to save several sets of data, each for a different application. SQL databases provide the ability for information to be transferred between one or more third-party applications easily. SQL Access Manager allows this data to be accessed and displayed in any InTouch application.

The InTouch SQL Access product consists of the SQL Access Manager program and the SQL Functions. The SQL Access Manager program is used to create and associate database columns with tagnames in your InTouch tagname dictionary. The process of associating database columns and InTouch database tagnames is called "binding." Binding the InTouch database tagnames to the database columns allows the SQL Access Manager to directly manipulate the data in the database. SQL Access Manager saves the database field names and their associations in a comma-separated variable (.CSV) formatted file named "SQL.DEF." (This file resides in the InTouch application directory and may be viewed or modified using SQL Access Manager or any text editor, such as Notepad.) The SQL Access Manager also creates Table Templates defining database structure and format.

For more information on Bind Lists and Table Templates, see Chapter 3, "Configuring SQL Access Manager."

SQL Functions can be used in any InTouch action script. These functions can be used to automatically execute based on operator input, a tagname value changing or when a particular set of conditions exist. For example, if an alarm condition exists, the application would execute a **SQLInsert()** or **SQLUpdate()** command to save all of the applicable data points and the state of the alarm. The SQL Functions can be used to create new tables, insert new records into tables, edit existing table records, clear tables, delete tables, select and scroll through records, etc.

Note Database systems not discussed in this user's guide are not supported.

About this Manual

This manual is divided into a series of logical building block chapters that describe the various aspects of using SQL Access Manager. It is written in a "procedural" format that tells you in numbered steps how to perform most functions or tasks.

If you are viewing this manual online, when you see text that is green, click the text to "jump" to the referenced section or chapter. When you jump to another section or chapter and you want to come back to the original section, a "back" option is provided.

Tip These are "tips" that tell you an easier or quicker way to accomplish a function or task.

The *InTouch User's Guide* will help you familiarize yourself with the WindowMaker development environment and its tools, read Chapter 1, "WindowMaker Program Elements." To learn about working with windows, graphic objects, wizards, ActiveX controls and so on, read Chapter 2, "Using WindowMaker."

For details on InTouch runtime environment (WindowViewer), see your online *InTouch Runtime User's Guide*.

In addition, the *InTouch Reference Guide* provides you with an in-depth reference to the InTouch script language, system tagnames, and tagname **.fields**.

For details on the add-on program, SPC Pro, see your *SPC Pro User's Guide*.

For details on the add-on program, Recipe Manager, see your *Recipe Manager User's Guide*.

Online manuals are also included in your FactorySuite software package for all FactorySuite components.

Note You must install the Adobe Acrobat Reader (version 4.0 or later) to view or print the online manuals.

Assumptions

This manual assumes you are:

- Familiar with the Windows 2000, Windows XP, and/or Windows NT operating system working environment.
- Knowledgeable of how to use of a mouse, Windows menus, select options, and accessing online Help.
- Experienced with a programming or macro language. For best results, you should have an understanding of programming concepts such as variables, statements, functions and methods.

Technical Support

Wonderware Technical Support offers a variety of support options to answer any questions on Wonderware products and their implementation.

Prior to contacting technical support, please refer to the relevant chapter(s) in your *SQL Access Manager User's Guide* for a possible solution to any problem you may have with your system. If you find it necessary to contact technical support for assistance, please have the following information available:

1. Your software serial number.
2. The version of InTouch you are running.
3. The type and version of the operating system you are using. For example, Microsoft Windows NT Version 4.0 workstation.
4. The exact wording of system error messages encountered.
5. Any relevant output listing from the Wonderware Logger, the Microsoft Diagnostic utility (MSD), or any other diagnostic applications.
6. Details of the attempts you made to solve the problem(s) and your results.
7. Details of how to recreate the problem.
8. If known, the Wonderware Technical Support case number assigned to your problem (if this is an on-going problem).

For more information on Technical Support, see your online *FactorySuite System Administrator's Guide*.

ODBC Compliant

SQL Access Manager is an ODBC compliant application that communicates with any database system, provided the database system has an ODBC driver available for it. Before you can use an ODBC driver, it must be configured via the Microsoft ODBC Administrator program to set up the links between the ODBC compliant application and the database.

To configure an ODBC driver

1. Run the Microsoft ODBC Administrator program.

2. Select a driver or data source, and then click **Add New Name**, **Set Default** or **Configure**. The **ODBC Driver Setup** dialog box.

Option	Description
Data Source Name	User-defined name that identifies the data source.
Description	User-defined description of this data source.
Database Directory	Identify the directory that contains the database files. If none is specified, the current working directory is used.

Tip Enter any other information required to configure the selected driver.

3. Click **OK**.

Tip The driver writes the values of each field to the ODBC.INI file. These values are the default values of a connection to the data source. The default values can be changed by modifying the data source fields. Entries can be inserted manually in the appropriate data source section of the ODBC.INI file for any attribute that is not supported by the ODBC Driver Setup dialog box.

C H A P T E R 2

Configuring and Connecting Databases

SQL Access Manager supports databases developed in Oracle, Microsoft SQL Server, and Microsoft Access. Each database's requirements are unique and particular. This chapter includes separate sections for each database, describing how to configure the particular database for communication with SQL Access Manager.

Contents

Using Oracle 8.0

To communicate with Oracle 8.0

1. Verify that the Oracle OLEDB Provider (MSDAORA.DLL) exists on your InTouch client machine. This file is installed by MDAC, which is installed when you install InTouch.
2. Connect to Oracle by executing the SQLConnect() function in an InTouch action script.

   For more information on the usage of SQLConnect(), see Chapter 4, "Using SQL Functions."

SQLConnect() Format

The SQLConnect() function is used to connect to Oracle databases. The connection string used by the SQLConnect() function is formatted as follows:

```
SQLConnect(ConnectionId,"<attribute>=<value>;
   <attribute>=<value>;...");
```

The following describes the attributes used by Oracle:

Attribute	Value
Provider	MSDAORA
User ID	User name.
Password	Password.
Data Source	Oracle Server machine name

Example

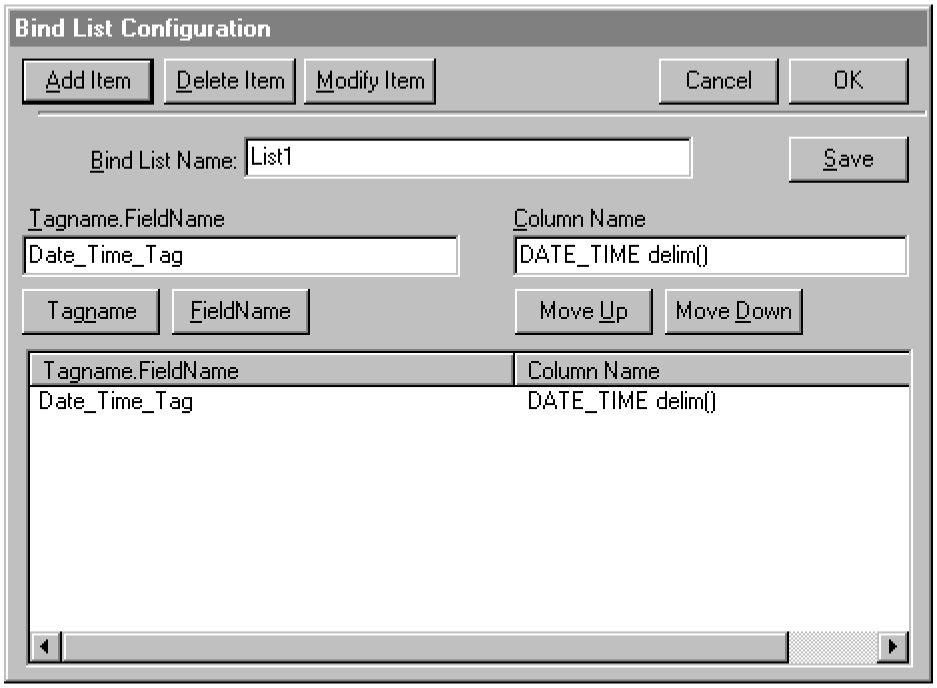
```
SQLConnect(ConnectionId,"Provider=MSDAORA; Data
   Source=OracleServer; User ID=SCOTT;
Password=TIGER;");
```

Logging Date and Time to an Oracle Date Field

To log the date and time to an Oracle date field, you must Configure the Bind List using the delim function.

To log both date and time to an Oracle date field

1. In the Application Explorer under **SQL Access Manager**, double-click **Bind List**. The **Bind List Configuration** dialog box appears.



Bind List Name: List1

Tagname.FieldName: Date_Time_Tag

Column Name: DATE_TIME delim()

Tagname.FieldName	Column Name
Date_Time_Tag	DATE_TIME delim()

2. In the **Tagname.FieldName** box, type the tagname that you want to use.
3. In the **Column Name** box, type the **DATE_TIME delim()** function.
4. In your InTouch application, create a QuickScript to prepare input data from present date and time. For example:

```
DATE_TIME_TAG = "TO_DATE('" + $DateString + "" +
   StringMid($TimeString,1,8) + "','mm/dd/yy
   hh24:mi:ss')";
```

Tip The Date_Time_Tag will display as the following in runtime:

```
TO_DATE('08/22/97 23:32:18' ,'mm/dd/yy hh24:mi:ss')
```

Using Microsoft SQL Server

To communicate with Microsoft SQL Server

1. Configure the Windows database client.
2. Connect to Microsoft® SQL Server by executing the **SQLConnect()** function in an InTouch QuickScript.

   For more information on the usage of SQLConnect(), see Chapter 4, "Using SQL Functions."

Configuring the Client

SQLConnect() Format

The **SQLConnect()** function is used to connect to Microsoft SQL Server. Executing this function logs you onto the database server and opens a connection to allow other SQL functions to be executed. The connection string used by the **SQLConnect()** function is formatted as follows:

```
SQLConnect(ConnectionId,"<attribute>=<value>;
   <attribute>=<value>;...");
```

The following describes the attributes used by Microsoft SQL Server:

Attribute	Value
Provider	SQLOLEDB
DSN	The name of the data source as configured in Microsoft ODBC Administrator.
UID	Logon ID, case sensitive.
PWD	Password, case sensitive.
SRVR	Name of the server computer with the database tables to be accessed.
DB	The database name to be accessed.

Example

```
SQLConnect(ConnectionId,"DSN=SQL_Data;UID=OPERATOR;PWD=XYZ
   Z");
```

Data Types Supported

SQL Access Manager associates the four data types in InTouch (discrete, integer, real, and message) with other data types in database systems. The char data type contains fixed length character strings. InTouch Message tagnames require a char data type. A field length must be specified. Microsoft SQL Server databases support a char field with a maximum length of 8,000 characters. However, InTouch Message tagnames are limited to 131 characters. If a message variable contains more characters than the length specified for a database field, the string will be truncated when inserted into the database**.**

The int data type represents InTouch Integer tagnames. If a field length is not specified, the length is set to the default value of the database. If the length is specified, it will be in the form Width. The Width determines the maximum number of digits for the column.

The float data type represents InTouch Real tagnames. The field length setting is fixed by the database. A field length for this data type is not required.

Using Microsoft Access

To communicate with Microsoft Access, you must connect to it by executing the **SQLConnect()** function in an InTouch QuickScript.

SQLConnect() Format

The **SQLConnect()** function is used to connect to Microsoft Access databases. Executing this function logs you on to the database server and opens a connection to allow other SQL functions to be executed. The connection string used by **SQLConnect()** is formatted as follows:

```
SQLConnect(ConnectionId,"<attribute>=<value>;
   <attribute>=<value>;...");
```

DSN is an attribute used by Microsoft Access and is the name of the data source as configured in the Microsoft ODBC Administrator.

Example

```
SQLConnect(ConnectionId,"DSN=MSACC");
```

String Length

The valid data types that SQL Access Manager supports depends on the version of the ODBC driver being used. The text data type contains fixed length character strings and are used with InTouch Message tagnames. A length must be specified. Microsoft Access databases support text fields with a maximum length of 255 characters. InTouch Message tagnames are limited to 131 characters. If a message variable contains more characters than the length specified for a database field, the string will be truncated when inserted into the database. The Microsoft Access ODBC driver supports up to 17 characters per column name. The maximum number of columns supported when using SQLSetStatement(Select Col1, Col2, ...) is 40.

Data Type Values for Supported Databases

Oracle

Data Type	Length	Default	Range	Tag Type
char	2,000 characters	1 character		Message
number	38 digits	38 digits		Integer

Microsoft SQL Server

Data Type	Length	Default	Range	Tag Type
char	8,000 characters			Message
int			-2,147,483,648 to 2,147,483,647	Integer
float	15 digits		-1.79E^{+308} to 1.79E^{+308}	Real

Microsoft Access 2000

Data Type	Length	Default	Range	Tag Type
text	255 characters			Message
number				Integer
number				Real

C H A P T E R 3

Configuring SQL Access Manager

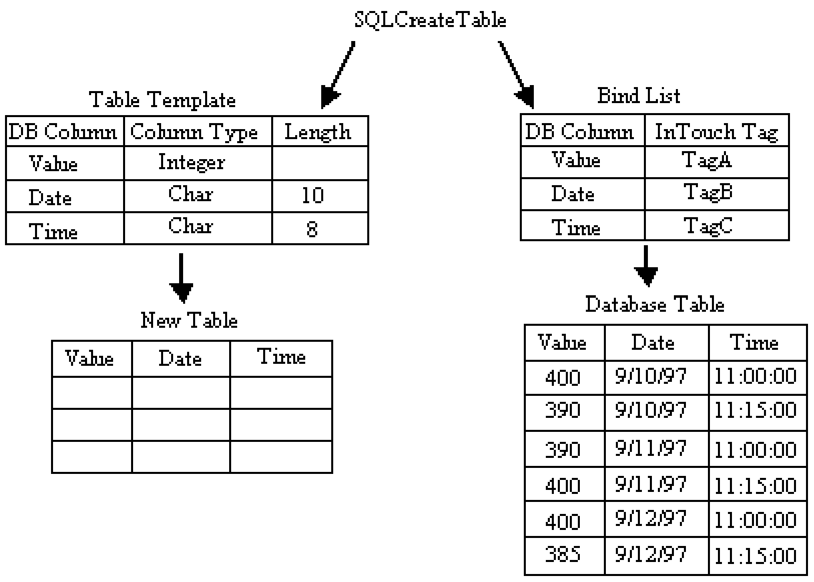
The SQL Access Manager utility program creates Bind Lists and Table Templates. The Bind List associates database columns with tagnames in the InTouch Tagname Data Dictionary. The Table Template defines the structure and format of a new table in the database.

Contents

SQL Access Manager Overview

When an InTouch application executes a **SQLCreateTable()** command, the Table Template argument defines the structure of the new database file.

When a **SQLInsert()**, **SQLSelect()** or **SQLUpdate()** is executed, the Bind List argument defines which InTouch tagnames are used and which database columns to associate.



DB Column	Column Type	Length
Value	Integer	
Date	Char	10
Time	Char	8

DB Column	InTouch Tag
Value	TagA
Date	TagB
Time	TagC

Value	Date	Time

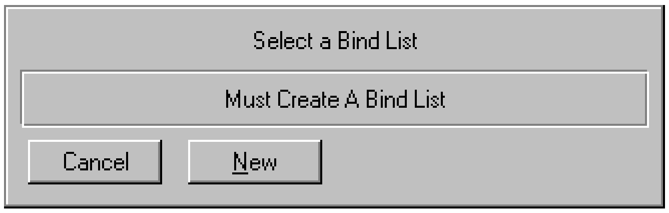
Value	Date	Time
400	9/10/97	11:00:00
390	9/10/97	11:15:00
390	9/11/97	11:00:00
400	9/11/97	11:15:00
400	9/12/97	11:00:00
385	9/12/97	11:15:00

Configuring a Bind List

The Bind List associates database columns with tagnames in the InTouch Data Dictionary.

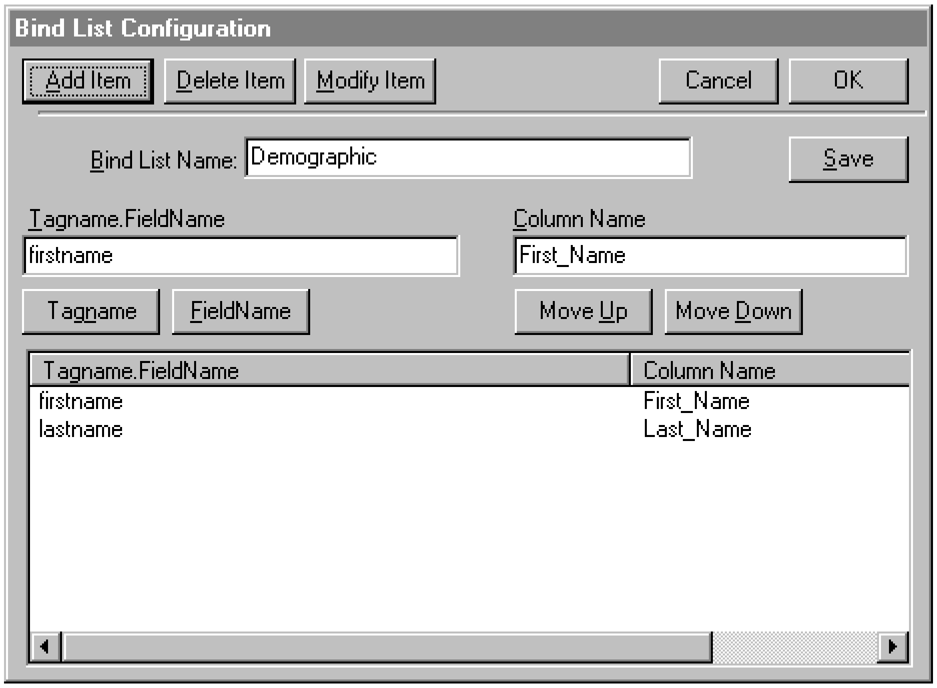
To create a new Bind List

1. On the **Special** menu, point to **SQL Access Manager**, and then click **Bind List**, or in the Application Explorer under **SQL Access Manager**, double-click **Bind List**.



2. Click **New.**

3. The **Bind List Configuration** dialog box appears.



Bind List Name: Demographic

Tagname.FieldName: firstname

Column Name: First_Name

Tagname.FieldName	Column Name
firstname	First_Name
lastname	Last_Name

Tip If you right click the mouse in any of the text entry boxes, a menu appears displaying the commands that you can apply to the selected text.

4. In the **Bind List Name** box, type the Bind List Name.

Tip A Bind List Name can be up to 32 characters in length. The new Bind List links database columns to InTouch tagnames. For example, if an employee demographic list is being created, you would enter the Bind List Name that associates information on the employees here.

Note The **SQLInsert()**, **SQLSelect()**, and **SQLUpdate()** functions use the Bind List parameter.

5. In the **Tagname.FieldName** box, type an InTouch **tagname.field** name.

Tip The Tagname Dictionary associates this **tagname.field** with the **Column Name** in the database. If this tagname is not currently defined in the Tagname Dictionary, double-click it to open the **Tagname Dictionary** dialog box and define it now.

6. Click **Tagname** to select an existing tagname. The Tag Browser appears.

Tip The Tag Browser will display the tagnames for the currently selected tag source. To select a tagname, double-click it or select it, and then click **OK**. To select a **.field** for the tagname click the **Dot Field** arrow, and select the **.field** that you want to use, and then click **OK**.

Note I/O type tagnames that are not used in your application, but are specified in a SQLAccess bind list, will be activated (advised to the I/O Server) as soon as WindowViewer starts up. No connect to a database is necessary to see this behavior.

For more information on the Tag Browser, see your online *InTouch User's Guide*.

7. Click **FieldName** to append a **.field** to the tagname. The **Choose field name** dialog box appears.
8. Click the **.field** that you want to use. The dialog box will close and the **.field** will automatically be appended to the tagname in the **Tagname.FieldNam**e field.

   For more information on tagname **.fields**, see Chapter 4 in your *InTouch User's Guide*.
9. In the **Column Name** box, type the name of the column.

Tip A Column Name can be up to 30 characters in length. The column name is directly associated with the database column name. If the Column Name has a space, use square brackets around the Column Name in the Bind List and when used in a script. For example:

```
WHERE EXPR= "[Pipe Flow} = " + text (tagname,"#");
```

Tip Special Delimiters can also be used to associate your column name with your database.

For more information on special delimiters, see "Using Special Delimiters"

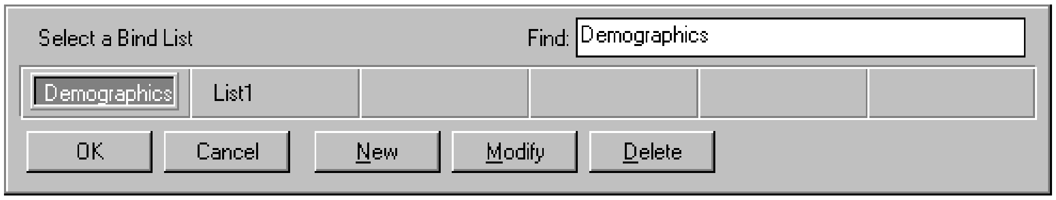
10. Click **Move Up** to move the selected tagname up one level in the list.
11. Click **Move Down** to move the selected tagname down one level in the list.
12. Click **Add Item** to add your new **Tagname.FieldName** and **Column Name** to the Bind List.
13. Click **Delete Item** to delete a selected **Tagname.FieldName** and **Column Name** from the Bind List.
14. Click **Modify Item** to modify a selected **Tagname.FieldName** or **Column Name** for this Bind List.
15. Click **OK** to save your new Bind List configuration and close the dialog box.

Tip You can click **Save** to save your settings without closing the dialog box.

To modify a Bind List

1. On the **Special** menu, point to **SQL Access Manager**, and then click **Bind List**, or in the Application Explorer under **SQL Access Manager**, double-click **Bind List**.

2. The **Select a Bind List** dialog box appears.

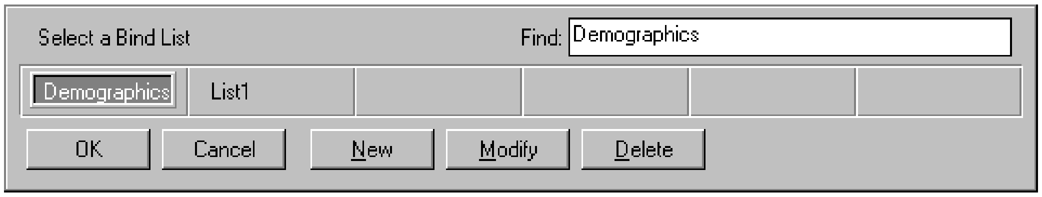


3. Select the Bind List name that you want to change, and then click **Modify**. The **Bind List Configuration** dialog box appears.
4. Modify the required item(s).
5. Click **OK** to save your changes and close the dialog box.

For more information on configuring a Bind List, see "To create a new Bind List."

To delete a Bind List

1. On the **Special** menu, point to **SQL Access Manager**, and then click **Bind List**, or in the Application Explorer under **SQL Access Manager**, double-click **Bind List**.
2. The **Select a Bind List** dialog box appears.



3. Select the Bind List name that you want to delete.
4. Click **Delete**. A message box appears asking you to confirm your deletion. Click **Yes** to delete the selected Bind List, or click **No** to cancel the deletion. The **Bind List Configuration** dialog box reappears.
5. Click **OK** to close the dialog box.

Using Special Delimiters

The **SQLInsert()** and the **SQLUpdate()** functions use a default format that encloses message strings with single quotes. Some SQL databases expect to receive message strings enclosed by another type of delimiter. For example, Oracle expects to receive a date string surrounded by brackets. When this occurs, the **Delim()** function must be used as follows:

In the **Bind List Configuration** dialog box **Column Name** field, after the column name, type the keyword "delim" (not case sensitive). The keyword "delim" must be entered followed by:

- a left parenthesis
- the left delimiter
- a comma

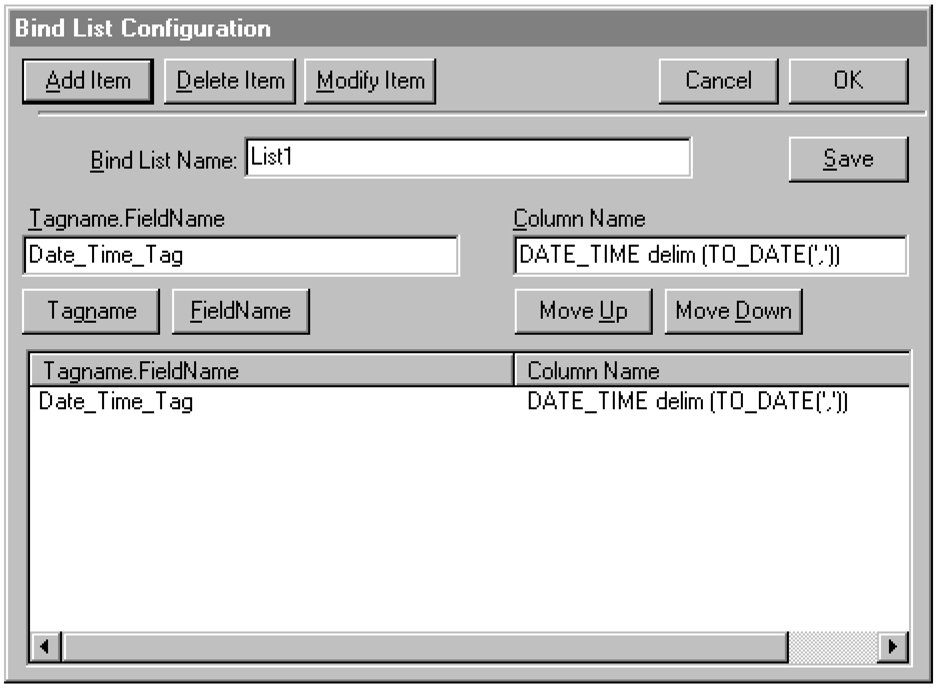
- the right delimiter
- a right parenthesis

Example: **datestring delim (‘,‘)**

To use the same delimiter for both left and right, just specify the delimiter in parentheses without the comma.

Example: **datestring delim (‘ ‘)**

The following example uses different left and right delimiters. Notice where **date delim (‘,’)** is entered in the **Column Name** field.



Bind List Name: List1

Tagname.FieldName: Date_Time_Tag

Column Name: DATE_TIME delim (TO_DATE(','))

Tagname.FieldName	Column Name
Date_Time_Tag	DATE_TIME delim (TO_DATE(','))

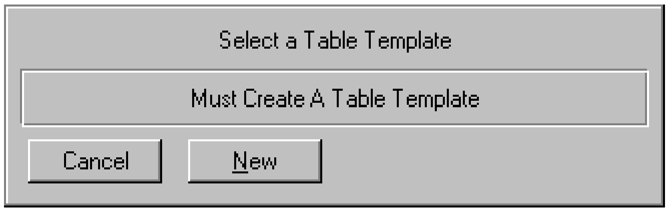
For more information on logging date and time to an Oracle date field, see Chapter 2, "Configuring and Connecting Databases."

Configuring a Table Template

This command creates a Table Template defining the structure and format of a new table in the database.

To create a new Table Template

1. On the **Special** menu, point to **SQL Access Manager**, and then click **Table Template**, or in the Application Explorer under **SQL Access Manager**, double-click **Table Template**.



2. Click **New**.
3. The **Table Template Configuration** dialog box appears.

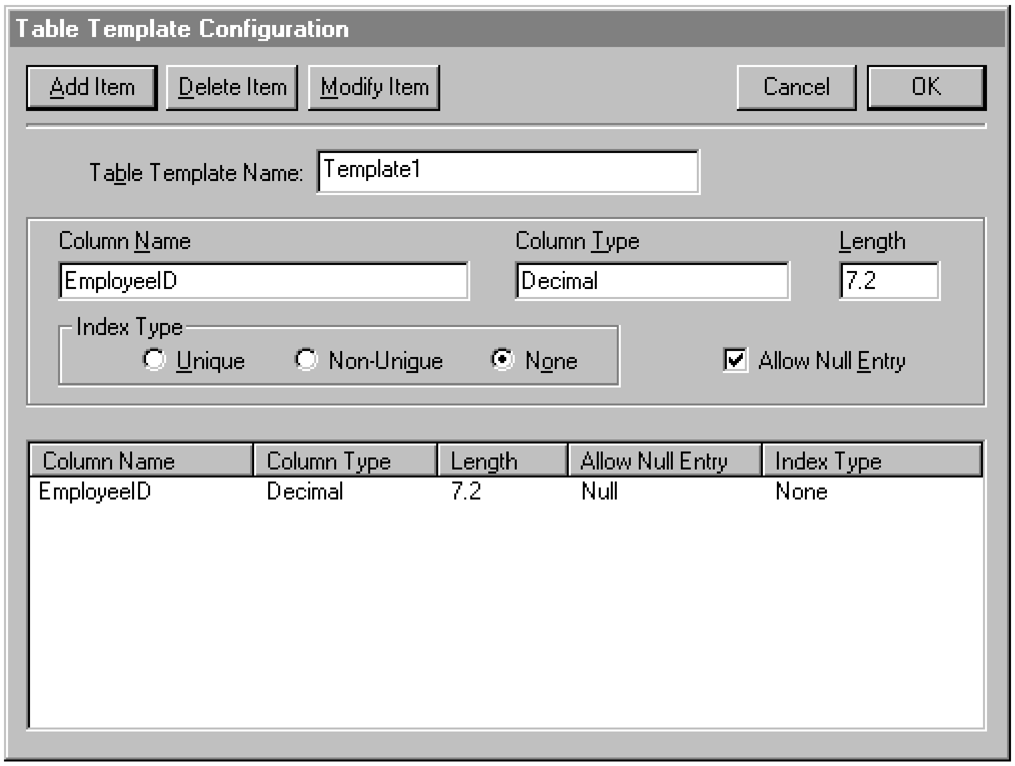


Table Template Name: Template1

Column Name: EmployeeID
Column Type: Decimal
Length: 7.2
Index Type: Unique Non-Unique [✓ None]
[✓ Allow Null Entry]

Column Name	Column Type	Length	Allow Null Entry	Index Type
EmployeeID	Decimal	7.2	Null	None

Tip If you right click the mouse in any of the text entry boxes, a menu appears displaying the commands that you can apply to the selected text.

4. In the **Table Template Name** box, type the name of the Table Template.

Note A Table Template Name can be up to 32 characters in length. If you are creating an index, unique or otherwise, the Table Template Name can not exceed **24 characters**. The Table Template name is used to identify the structure of a database for the **SQLCreateTable()** function.

5. In the **Column Name** box, type the column name for the Table Template. A Column Name can be up to 30 characters in length.

6. In the **Column Type** box, type the data type for the column. Data type selections vary according to the database being used.

   For more information on data types for a specific database, see Chapter 2, "Configuring and Connecting Databases."

7. Select the **Index Type** as follows:

   Unique
   A column requires that each value in that column be unique.

   Non-Unique
   A column does not require that each value in that column be unique.

   None
   No Index.

   Tip When you execute a **SQLCreateTable()**, an index file is automatically created.

8. Select **Allow Null Entry** to allow null data to be entered in this column.

   Note InTouch does not support null data.

   When inserting data, if a value has not been entered for a tagname, values will be:

Data Type	Value
Discrete	0
Integer	0
Message	Strings with no characters

   When selecting data, null values will be translated according to the data type as shown above.

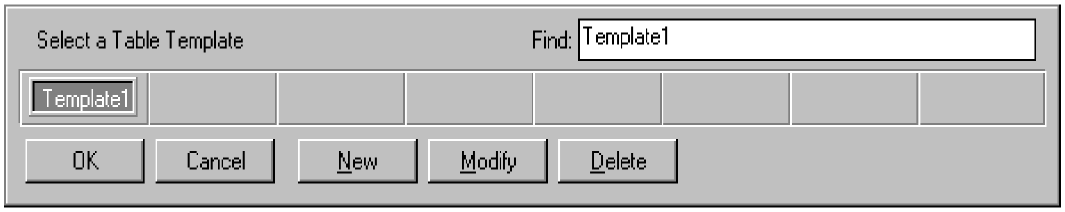
9. Click **Add Item** to add your new Column Name, Column Type, Length and Index Type to the Table Template.
10. Click **Delete Item** to delete a selected Column Name, Column Type, Length and Index Type from the Table Template list.
11. Click **Modify Item** to modify a selected Column Name, Column Type, Length and Index Type in the Table Template list.
12. Click **OK** to save your new Table Template configuration and close the dialog box.

   Tip You can click **Save** to save your settings without closing the dialog box.

To modify a Table Template

1. On the **Special** menu, point to **SQL Access Manager**, and then click **Table Template**, or in the Application Explorer under **SQL Access Manager**, double-click **Table Template**.

2. The **Select a Table Template** dialog box appears.

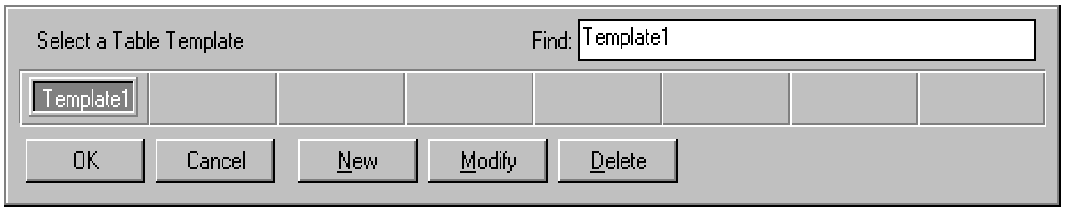


3. Select the Table Template name that you want to modify, and then click **Modify**. The **Table Template Configuration** dialog box appears.
4. Modify the required item(s).
5. Click **OK** to save your changes and close the dialog box.

   For more information on configuring a Table Template, see "To create a new Table Template."

To delete a Table Template

1. On the **Special** menu, point to **SQL Access Manager**, and then click **Table Template**, or in the Application Explorer under **SQL Access Manager**, double-click **Table Template**.
2. The **Select a Table Template** dialog box appears.



3. Select the Table Template name that you want to delete
4. Click **Delete**. A message box appears asking you to confirm your deletion. Click **Yes** to delete the selected Bind List, or click **No** to cancel the deletion. The **Table Template Configuration** dialog box reappears.
5. Click **OK** to close the dialog box.

The SQL.DEF File

The SQL Access Manager saves the configuration information for the Bind Lists and Table Templates to a file named "SQL.DEF." This file is formatted as a comma-separated variable (.CSV) type file. The SQL.DEF file can be viewed or modified using SQL Access Manager or any text editor, such as Notepad. The data appears in the file as follows:

:BindListName,*BindListName*

Tagname1.FieldName,ColumnName1

Tagname2.FieldName,ColumnName2

Tagname3.FieldName,ColumnName3

:TableTemplateName,*TableTemplateName*

ColumnName1,ColumnType,[ColumnLength],Null,Index

ColumnName2,ColumnType,[ColumnLength],Null,Index

ColumnName3,ColumnType,[ColumnLength],Null,Index

C H A P T E R 4

Using SQL Functions

InTouch uses SQL Functions to interact with information in the database. These functions are an extension of the standard InTouch QuickScript functions and can be used in any script. They allow you to select, modify, insert or delete records in the tables you choose to access.

Contents

SQL Functions

This section lists each SQL Function. Keep in mind that SQL actions are synchronous. Control is not returned to InTouch until the SQL activity is complete (InTouch trending, polling, etc. are suspended).

All SQL Functions (with the exception of **SQLNumRows()**) return a *ResultCode*. If the *ResultCode* is non-zero, the function failed and other actions should be taken. The *ResultCode* can be used by the **SQLErrorMsg()** function.

The general format for SQL Functions is as follows:

```
SQLFunction(Parameter1, Parameter2,...)
```

For complete details on each SQL function and examples of how you use each function, see your *InTouch Reference Guide.*

Function

SQLAppendStatement(ConnectionId, SQLStatement)

Append the statement SQLStatement to the default SQL statement for *ConnectionId.*

SQLClearParam(StatementId, ParameterNumber)

Set the value of *ParameterNumber* associated with StatementId to zero or a zero-length string, depending on whether the parameter is of numeric or string type.

SQLClearStatement(ConnectionId, StatementId)

Clean up resources associated with StatementId. However, the default statement associated with *ConnectionId* remains intact.

SQLClearTable(ConnectionId, TableName)

Delete all rows in the table named TableName.

SQLCommit(ConnectionId)

Commit the transaction that was created by the last SQLTransact.

SQLConnect(ConnectionId, ConnectString)

The *ConnectString* parameter is the same ConnectionString as explained in most ADO documentations (probably most extensively by Microsoft ADO API Reference). It is a parameter that may need to be modified in any InTouch application to leverage the power of native OLE DB provider for a particular database management system.

A general form of the *ConnectString* parameter consists of different components separated by semicolons. The first component is normally specified as Provider=ProviderName, where ProviderName is the OLE DB provider for the particular database system. The SQLConnect functions in existing InTouch applications do not have the Provider keyword in the *ConnectString* parameter, thus ADO will use the default provider, Microsoft OLE DB Provider for ODBC, which is MSDASQL.DLL. Although existing InTouch applications will continue to work, it is highly recommended that the *ConnectString* parameter be changed to use the native OLE DB provider. Examples of ConnectString include the following:

Example 1

Microsoft OLE DB Provider for Microsoft Jet (recommended use)

"Provider=Microsoft.Jet.OLEDB.4.0;Data Source=d:\DBName.mdb;User ID=UserIDStr;Password=PasswordStr;"

Microsoft.Jet.OLEDB.4.0 is the native OLE DB Provider for Microsoft Jet (Microsoft Access Database engine).

Example 2

Microsoft OLE DB Provider for ODBC (using the default provider MSDASQL for MS Access):

"Provider=MSDASQL;DSN=DSNStr;UID=UserName;PWD=PasswordStr;"

Note User ID and uid can be used interchangeably, and Password and pwd can be used interchangeably. However, as stated above, it is recommended that the ConnectString parameter uses Microsoft.Jet.OLEDB.4.0.

Example 3

Microsoft OLE DB Provider for SQL Server (recommended use)

"provider=sqloledb;Data Source=MyServer;Initial Catalog=MyDB;User Id=sa;Password=;"

The OLE DB Provider for SQL Server is sqloledb.

Example 4

Microsoft OLE DB Provider for SQL Server (recommended use)

"Provider=SQLOLEDB;uid=sa;pwd=;Database=MyDB"

Example 5

Microsoft OLE DB Provider for ODBC (using the default provider MSDASQL for SQL Server):

"DSN=Pubs;UID=sa;PWD=;"

Example 6

Microsoft OLE DB Provider for ODBC (using the default provider MSDASQL for SQL Server):

"Data Source=Pubs;User ID=sa;" "Password=;"

Note Data Source and Server can be used interchangeably, and Initial Catalog and Database can be used interchangeably.

Example 7

Microsoft OLE DB Provider for Oracle (recommended use)

"Provider=MSDAORA;Data Source=ServerName;User ID=UserIDStr; Password=PasswordStr;"

If SQLTrace=1 is defined under the [InTouch] section of the win.ini file, each successful execution of SQLConnect will log version information for the ADO, the provider and the database system to the Wonderware logger.

SQLCreateTable(ConnectionId, TableName, TemplateName)

Create a table named TableName using the TemplateName.

SQLDelete(ConnectionId, TableName, WhereExpr)

Delete the rows that match the *WhereExpr* clause from *TableName*.

SQLDisconnect(ConnectionId)

Disconnect from the database and clean up all resources that were created by SQLPrepareStatement and SQLInsertPrepare that have not yet been released (by executing SQLClearStatement and SQLInsertEnd).

SQLDropTable(ConnectionId, TableName)

Delete the table named TableName from the database.

SQLEnd(ConnectionId)

Clean up resources associated with the logical table associated with ConnectionId.

SQLErrorMsg(ResultCode)

Return a *ResultCode* of -1 whenever an error is generated by the database provider. The *ResultCode* returned is always -1, but the message is copied exactly from the provider.

For a list of Result Codes and a description of the error messages, see Chapter 5, "Troubleshooting."

SQLExecute(ConnectionId, BindList, StatementId)

Execute the statement associated with *StatementId* (MS Access query, MS SQL Server stored procedure, or a textual SQL statement). The *BindList* parameter can be a zero-length string. If *StatementId* is associated with a row-returning query, then the logical table is updated with the result of SQLExecute. If a real bind list is specified, then the result is associated with the *BindList*. A zero-length BindList is useful when it is known in advance that the *StatementId* is not associated with a row-returning query.

SQLFirst(ConnectionId)

Go to the first row of the logical table and fetch values of that row into InTouch tags.

SQLGetRecord(ConnectionId, RecordNumber)

Go to row number *RecordNumber* of the logical table and fetch values of that row into InTouch tags.

SQLInsert(ConnectionId, TableName, BindList)

Use the current values of InTouch tags to insert one row into *TableName*.

SQLInsertEnd(ConnectionId, StatementId)

Clean up resources associated with *StatementId* created by SQLInsertPrepare.

SQLInsertExecute(ConnectionId, BindList, StatementId)

Use the current values of InTouch tags to insert one row into the table identified by the previous SQLInsertPrepare. If the *BindList* includes an Identity key field for a MS SQL Server table, it is necessary to set the IDENTITY_INSERT option before running SQLInsertExecute.

Example

Inserting a row with an identity key that is part of a BindList:

```
ResultCode = SQLSetStatement(ConnectionId, "SET
   IDENTITY_INSERT Products ON");
ResultCode = SQLExecute(ConnectionId, "", 0);
ResultCode = SQLInsertPrepare(ConnectionId, TableName,
   Bindlist, StatementId);
ResultCode = SQLInsertExecute(ConnectionId, Bindlist,
   StatementId);
ResultCode = SQLInsertEnd(ConnectionId, StatementId);
```

SQLInsertPrepare(ConnectionId, TableName, BindList, StatementId)

Return a *StatementId* to be used in SQLInsertExecute and SQLInsertEnd.

SQLLast(ConnectionId)

Go to the last row of the logical table and fetch values of that row into InTouch tags.

SQLLoadStatement(ConnectionId, FileName)

Load the statement contained in the file *FileName* into the default statement for *ConnectionId*.

SQLManageDSN(ConnectionId)

The *ConnectionId* is not used. It is retained for backward compatibility of older versions of SQL Access. Therefore, any number can be passed into the function. No database connection needs to be established before this function can be called.

Example

```
SQLManageDSN( 0 )
```

SQLNext(ConnectionId)

Go to the next row of the logical table and fetch values of that row into InTouch tags.

SQLNumRows(ConnectionId)

Return the number of rows of the logical table. Because this function may return an error code, the recommended use of the function is as follows:

```
DIM TEMP AS INTEGER;

TEMP = SQLNumRows(ConnectionId);

IF (TEMP >= 0) THEN
   RowCount = TEMP;

ELSE
   ResultCode = TEMP;

ENDIF;
```

Definition

A default statement is a statement associated with a connection ID. It can be a textual SQL statement (SELECT, INSERT, DELETE, or UPDATE), the name of a query in MS Access (with or without parameters), or the name of a stored procedure in MS SQL Server (with or without parameters). The default statement is modified by SQLLoadStatement, SQLSetStatement and SQLAppendStatement and is used by SQLExecute whenever StatementId = 0 is specified.

SQLPrepareStatement(ConnectionId, StatementId)

Prepare the default statement and return a *StatementId* (1, 2, 3, and so on). This preparation is useful for statements with parameters that need to be set using the SQLSetParam*{Type}* functions. *SQLHandle* is specified as the second parameter to this function in older versions of SQL Access; however, the current version of SQL Access renames SQLHandle into StatementId for all functions. The functional behavior remains the same.

SQLPrev(ConnectionId)

Go to the previous row of the logical table and fetch values of that row into InTouch tags.

SQLRollback(ConnectionId)

Roll back the transaction that was created by the last SQLTransact.

SQLSelect(ConnectionId, TableName, BindList, WhereExpr, OrderByExpr)

Instructs the database to retrieve information from a table. When a **SQLSelect()** function is executed, a temporary Results Table is created in memory, containing records that can be browsed using **SQLFirst()**, **SQLLast()**, **SQLNext(), SQLNumRows** and **SQLPrev()**.

Execute the statement:

```
SELECT FROM TableName WHERE WhereExpr ORDER BY OrderByExpr
```

If the statement is executed successfully, a temporary record set (referred to as the logical table) is created and the *BindList* is used to associate InTouch tags with the columns of this table in preparation for SQLFirst, SQLPrev, SQLNext, SQLLast, and SQLNumRows. This logical table remains valid even if it has no row. For example, if *WhereExpr* is False for all records.

SQLSetParamChar(StatementId, ParameterNumber, Value, Length)

Set the parameter *ParameterNumber* associated with *StatementId* to a character string value (the string can be a single character). The last parameter to the function specifies the maximum length of the parameter. If the length of *Value* is longer than the length specified, *Value* will be truncated to the specified length. If length is specified as 0, the entire length of *Value* will be used.

SQLSetParamDate(StatementId, ParameterNumber, Value)

Set the parameter *ParameterNumber* associated with *StatementId* to a date value. The time is considered as 12:00:00 AM (the beginning of the date specified).

SQLSetParamDateTime(StatementId, ParameterNumber, Value, Precision)

Set the parameter *ParameterNumber,* associated with *StatementId,* to a date/time value.

SQLSetParamDecimal(StatementId, ParameterNumber, Value, Precision, Scale)

Set the parameter *ParameterNumber,* associated with *StatementId*, to a decimal value. *Value* can be either a string (or an InTouch message tag) that represents a decimal number (123.456) or a numeric value (or an InTouch memory real tag). It is recommended that a message tag is used instead of a real tag to guarantee the precision of the parameter. However, if *Value* must be a floating point number (for example, a real value received from an I/O server), then the function will continue to work, but high precision may not be guaranteed because of the limitation of floating point representation. *Precision* is the total number of digits in the number, and *Scale* is the number of digits to the right of the decimal point.

SQLSetParamFloat(StatementId, ParameterNumber, Value)

Set the parameter *ParameterNumber,* associated with *StatementId*, to a 64-bit, signed, floating-point value.

SQLSetParamInt(StatementId, ParameterNumber, Value)

Set the parameter *ParameterNumber,* associated with *StatementId*, to a 16-bit, signed, integer value.

SQLSetParamLong(StatementId, ParameterNumber, Value)

Set the parameter *ParameterNumber,* associated with *StatementId*, to a 32-bit, signed, integer value.

SQLSetParamNull(StatementId, ParameterNumber, Type, Precision, Scale)

Set the parameter *ParameterNumber,* associated with *StatementId*, to NULL.

The *Type* parameter can have the following value:

0: string

1: date/time

2: integer

3: float

4: decimal

Comparison with NULL value is controlled by the ANSI_NULLS option in MS SQL Server. The time of resolving this option depends on the database system. In SQL Server 7.0, this option is resolved at object creation time (not at query execution time). When a stored procedure is created in SQL Server 7.0, this option is ON by default and thus a clause such as "WHERE MyField = NULL" always returns NULL (FALSE) and no row is returned from a SELECT statement using this clause. In order for the comparison = or <> to return TRUE or FALSE, it is necessary to set the option to OFF when creating the stored procedure. If the ANSI_NULLS is not set to OFF, then SQLSetParamNull will not work as expected. In this case, comparison against NULL value should use the syntax "WHERE MyField IS NULL" or "WHERE MyField IS NOT NULL".

Example

Using SQLSetParamNull to return all rows in the Products table where the ProductName is not NULL.

Suppose a stored procedure is created in SQL Server using the following text.

```
SET ANSI_NULLS  OFF

GO

CREATE PROCEDURE sp_TestNotNull @ProductParam varchar(255)

AS SELECT * FROM Products WHERE ProductName <>
   @ProductParam

GO

SET ANSI_NULLS  ON
```

```
GO
```

InTouch can execute the following SQL Access scripts.

```
ResultCode = SQLSetStatement(ConnectionId,
   "sp_TestNotNull");

ResultCode = SQLPrepareStatement(ConnectionId,
   StatementId);

ResultCode = SQLSetParamNull(StatementId, 1, 0, 0, 0);

ResultCode = SQLExecute(ConnectionId, BindList,
   StatementId);

ResultCode = SQLFirst(ConnectionId);

ResultCode = SQLClearStatement(ConnectionId, StatementId);
```

SQLSetParamTime(StatementId, ParameterNumber, Value)

Set the parameter *ParameterNumber*, associated with *StatementId*, to a time value. The system current date is used along with the time specified.

SQLSetStatement(ConnectionId, SQLStatement)

Set the statement SQLStatement into the default SQL statement for ConnectionId.

SQLTransact(ConnectionId)

Begin a database transaction. Transactions can be nested as supported by the underlying OLE DB provider for the database system. For example, native OLE DB provider for Microsoft Jet supports transactions nested up to five levels, including the first and last transactions.

SQLUpdate(ConnectionId, TableName, BindList, WhereExpr)

Use the current values of InTouch tags to update all rows in the table named *TableName* matched by the *WhereExpr* clause.

SQLUpdateCurrent(ConnectionId)

Update the current row of the logical table using InTouch tags mapped to the table fields via the bind list specified in SQLSelect or SQLExecute. If there are rows that are identical to the current row, all of them will be updated. If there are too many identical rows to be updated in SQL Access, this function may return an error after updating a number of rows. The error message may be similar to, "Microsoft Cursor Engine: Key column information is insufficient or incorrect. Too many rows were affected by update." Up to 54 identical rows may be modified at once.

To avoid this situation, create a unique key field in the table so that no rows are identical. It is strongly recommended that all tables used by SQL Access have a unique key. For a table without a key, it is recommended that a field of type AutoNumber (MS Access) or an integer field used as the row Identity (SQL Server) be used as the primary key so that SQLUpdateCurrent affects only one row. This primary key field does not have to be included in a BindList.

SQL Parameters

The following describes the parameters required for each SQL function. When a parameter is entered in a script surrounded by quotation marks ("Parameter1") that exact string will be used. If no quotation marks are used, Parameter1 is assumed to be a tagname and the system will access the InTouch tagname dictionary for the value of the tagname, Parameter1.

Example

```
"c:\main\file" vs. Location
```

where: location is an InTouch message tagname

"c:\main\file" is a literal string

The parameters for most of the SQL functions will be one or more of the following:

BindList

Corresponds to one of the Bind List names in the SQL.DEF file.

ConnectionID

Memory integer tagname created by the user to hold the number (ID) assigned by the SQLConnect function to each database connection.

ConnectString

String that identifies the database and any additional logon information used in **SQLConnect()**.

ErrorMsg

Message variable containing a text description of the error message.

For more information on error message descriptions, see Chapter 5, "Troubleshooting."

FileName

The name of the file name in which the information is contained.

MaxLen

Maximum size of the column with which this parameter is associated. This setting determines whether the parameter is of varying character or long varying character type. If *MaxLen* is less than or equal to the largest character string allowed by the database, then the parameter is varying character type. If greater, long varying character type.

OrderByExpression

Defines the columns and direction for sorting. Only column names can be used to sort. The expression must be formatted:

ColumnName [ASC|DESC]

To sort the selected table by a column name (e.g., manager) in ascending order:

"manager ASC"

To sort by multi-columns, the expression is formatted:

ColumnName [ASC|DESC],
ColumnName [ASC|DESC]

To sort a selected table by one column name (for example, temperature) in ascending order and another column name (for example, time) in descending order:

"temperature ASC, time DESC"

ParameterNumber

Actual parameter number in the statement.

ParameterType

Data type of the specified parameter. Valid values:

Type	Description
Char	Blank Padded fixed length string
Var Char	Variable Length String
Decimal	BCD Number
Integer	4-byte signed integer
Small integer	2-byte signed integer
Float	4-byte floating point
Double Precision Float	8-byte floating point
DateTime	8-byte date time value
Date	4-byte date time value

Type	Description
Time	4-byte date time value
No Type	No Data Type

ParameterValue

Actual value to set.

Precision

Is the decimal value's precision, the max. size of the character, or the length in bytes of the date-time value.

RecordNumber

Actual record number to retrieve.

ResultCode

Integer variable returned from most SQL functions. *ResultCode* is returned as zero (0) if the function is successful and a negative integer if it fails.

For more information, see Chapter 5, "Troubleshooting."

Scale

Is the decimal value's scale. This value is required only if applicable to the parameter being set to null.

StatementId

When using the advanced functionality statements, SQL returns a *StatementId*, which it uses internally.

SQLStatement

Actual statement, for example:

```
ResultCode=SQLSetStatement(ConnectionID,”Select LotNo,
   LotName from LotInfo”);
```

TableName

The database table name you want to access.

TemplateName

The name of the template definition you want to use.

WhereExpression

Defines a condition that can be either true or false for any row of the table. The command extracts only those rows from the table for which the condition is true. The expression must be in the following format:
ColumnName *comparison_operator* expression

Note If the column is a character data type, the expression must be in single quotes.

The following example will select all rows whose name column contains the value EmployeeID:

```
name='EmployeeID'
```

The following example will select all rows containing part numbers from 100 to 199:

```
partno>=100 and partno<200
```

The following example will select all rows whose temperature column contains a value that is greater than 350:

```
temperature>350
```

Using SQL Functions in InTouch QuickScripts

SQL functions can be automatically inserted into InTouch QuickScripts by clicking on the **Add-ons** button within the QuickScript editor dialog. The SQL function will be automatically inserted into the script at the current cursor position.

For complete details on InTouch QuickScripts see your *InTouch User's Guide*, Chapter 6, "Creating QuickScripts in InTouch."

Specifying Complex Queries

SQL Access Manager allows you to specify complex queries and SQL statements of your own design. These queries may either be built dynamically or be contained in external files. Additionally, these queries may contain parameters that need to be "passed" into the query at runtime. These queries must then be executed and possibly have result sets returned. The SQL Access Manager API allows you to execute whatever SQL statement your database can handle and retrieve the result of that query. As a by-product, stored procedures are also available for execution by you. (Stored procedures are not fully supported.)

For more information on stored procedures, see "Supporting Stored Procedures."

Building Queries Dynamically

To build queries dynamically, two additional functions are required: **SQLSetStatement()** and **SQLAppendStatement()**. **SQLSetStatement()** starts a new SQL statement. This can be any valid SQL statement, including the name of a stored procedure. Since InTouch only supports character strings of 131 characters, **SQLAppendStatement()** is provided to concatenate additional strings onto the statement.

Note **Bold** text refers to SQL Query language commands.

Example

```
ResultCode = SQLSetStatement (ConnectionID, "Select LotNo,
   LotName, LotDescription, LotQuantity from LotInfo,
   ProductionInfo");
ResultCode = SQLAppendStatement (ConnectionID, " where
   LotInfo.LotNo = ProductionInfo.LotNo");
ResultCode = SQLAppendStatement (ConnectionID, " order by
   LotNo,NotName,LotQuantity");
```

The statement is now ready for execution.

Note Many database column and table names are case sensitive. For the above script to function properly, the column and database names must be typed exactly as used in the database tables.

Reading SQL Statements from a File

You can model your query in other packages such as, Microsoft Access and other 3rd party database tools, then use SQL Access for InTouch to perform your query. As several of these packages will generate the SQL statement, it's a simple matter to take that SQL statement and store it into a file by using the **SQLLoadStatement()**.

Example

```
ResultCode = SQLLoadStatement (ConnectionID,
   "c:\myappdir\lotquery.sql");
```

The statement is now ready for execution.

Modifying Extended SQL Statements

To provide full SQL functionality, SQL Access Manager allows you to specify a where clause that contains a value of an InTouch tagname. To allow runtime specification of SQL parameters, the following functions are provided:

- SQLPrepareStatement()
- SQLSetParam*Type*()
- SQLClearStatement()
- SQLClearParam()

To perform parameter substitution on a SQL statement, put a "?" in the SQL statement where you want to specify a parameter at a later date. The statement is "prepared," parameters are "set" into the statement, and then the statement is executed.

SQLPrepareStatement() prepares the statement for execution. It does not execute the statement, it just makes the statement active so you can set parameter values. **SQLSetParamType()** is a set of functions that allow you to set values into parameters in the SQL statement.

Example

```
ResultCode = SQLSetStatement (ConnectionID, "Select LotNo,
   LotName, LotDescription, LotQuantity from LotInfo,
   ProductionInfo");

ResultCode = SQLAppendStatement (ConnectionID, " where
   LotInfo.LotNo = ?");

ResultCode = SQLAppendStatement (ConnectionID, " order by
   LotNo,NotName,LotQuantity");

ResultCode = SQLPrepareStatement (ConnectionID,
   StatementId);

{return the statement handle into tag 'StatementId'}

ResultCode = SQLSetParamInt (StatementId, 1,
   tagLotNumber);

{put the value of tagLotNumber into param}
```

Since the statement only has one parameter, it is now ready for execution.

Once the statement is executed and you are finished with the prepared statement, **SQLClearStatement()** can be called to free the resources associated with that statement.

Note **SQLEnd()** is called to free "unnamed" SQL statements (those generated by existing SQL Access functions), and those statements created by **SQLSetStatement()** and **SQLLoadStatement()** and not prepared.

Executing Extended SQL Statements

Now that the statement has been either built dynamically or read from a file, and has been optionally prepared and modified, it's time to execute it. The SQL Access Manager API uses the **SQLExecute()** function to accomplish this. **SQLExecute()** will either execute the currently active statement (i.e., the one created by **SQLSetStatement()** or **SQLLoadStatement()**) or the statement that has been previously prepared and is specified by the statement handle parameter.

Example 1

```
ResultCode = SQLLoadStatement (ConnectionID,
   "c:\myappdir\lotquery.sql");

ResultCode = SQLExecute (ConnectionID, "BindList", 0);

{put the results of the select into the tags specified in
   BindList. prepared statement handle is zero}

ResultCode = SQLNext (ConnectionID);
```

```
{Get results of Select}
```

Example 2

```
ResultCode = SQLSetStatement (ConnectionID, "Select LotNo,
   LotName, LotDescription, LotQuantity from LotInfo,
   ProductionInfo");

ResultCode = SQLAppendStatement (ConnectionID, " where
   LotInfo.LotNo = ?");

{question mark means I'll  get back to you}

ResultCode = SQLAppendStatement (ConnectionID, " order by
   LotNo,NotName,LotQuantity");

ResultCode = SQLPrepareStatement (ConnectionID,
   StatementId);

{return the statement handle into tag 'StatementId'}

ResultCode = SQLSetParamInt (StatementId, 1,
   tagLotNumber);

{put the value of tagLotNumber into param}

ResultCode = SQLExecute (ConnectionID, "BindList",
   StatementId); {put the results of the Select into the
   tags specified in

BindList prepared statement handle is in StatementId}

ResultCode = SQLNext (ConnectionID);

{Get results of Select}
```

Example 3

SQLSetStatement – This statement must be used for complex queries and string expressions greater than 131 characters. When the string expression exceeds 131 characters use the SQLAppend

```
SQLSetStatement(ConnectionID, “Select  Speed, Ser_No from
   tablename where Ser_No =’” +   Serial_input  + “’”);

SQLExecute(ConnectionID, "BindList", 0);
```

In the above example the StatementId is set to zero so the statement does not have to call SQLPepare(Connection_Id, StatementId) before the execute statement. Because the StatementId was not created by the SQLPepare to properly end this select use the SQLEnd function instead of the SQLClearStatement().

```
SQLSetStatement(Connection_Id, “Select  Speed, Ser_No from
   tablename where Ser_No =’” +   Serial_input  + “’”);

SQLPrepareStatement(Connection_Id, StatementId);

SQLExecute(Connection_Id, StatementId);
```

In the above example the StatementId is created by the SqlPrepareStatement and used in the SQLExecute function. To end this select statement use SQLClearStatement to free up resources and free the StatementId.

Supporting Stored Procedures

The **SQLExecute()** function supports the execution of some stored procedures. For example, suppose you create a stored procedure on the database server named "LotInfoProc," that contains the following select statement: "Select LotNo, LotName from LotInfo." You would write the following InTouch QuickScript to execute the procedure and get the results:

Using Microsoft SQL Server

```
ResultCode = SQLSetStatement (ConnectionID,
   "LotInfoProc");

ResultCode = SQLExecute(ConnectionID, "BindList", 0);

ResultCode = SQLNext (ConnectionID);

{Get results of Select}
```

Using Oracle or Microsoft Access

```
ResultCode = SQLSetStatement (ConnectionID, "{CALL
   LotInfoProc}");

ResultCode = SQLExecute(ConnectionID, "BindList", 0);

ResultCode = SQLNext (ConnectionID);

{Get results of Select}
```

Fetching Values into InTouch Tags

The five script functions SQLFirst, SQLPrev, SQLNext, SQLLast, and SQLGetRecord allow navigating among rows of the logical table and fetching field values into InTouch tags. If a field is NULL, the value of the associated InTouch tag will be a zero or a zero-length string depending on whether the tag is of analog or message type. If a string in the database is greater than 131 characters, only the first 131 characters are copied into the associated InTouch message tag.

Persisting InTouch Tags into Database Field Values

The four script functions SQLUpdate, SQLUpdateCurrent, SQLInsert, and SQLInsertExecute allow updating or inserting into a table using InTouch tag values. If an InTouch message tag is longer than the defined size of the corresponding text field of the table, the number of characters used from the message tag will be the defined size of the field. Since InTouch tags cannot be NULL, it is impossible to update or insert NULL values into the database using these functions if the BindList includes the field. The way to insert NULL values into a field is to use SQLExecute on an INSERT statement that does not include the field, which should have been defined to allow NULL values.

Implications of the Data Updating Rules

The rules for fetching values into InTouch tags and persisting data into table fields imply that it is possible to modify values in the table unintentionally in the following scenarios.

Unintentional Conversion of NULL Values into Zeros or Empty Strings

Execution of one of the navigation functions fetches NULL values into InTouch tags as zeros or zero-length strings (e.g. Tag1). After some other tags in teh BindList are updated, execution of SQLUpdateCurrent persists the zeros or zero-length strings back to the table, overwriting the NULL value associated with Tag1. Execution of SQLUpdate will update rows using these zeros or zero-length strings from Tag1 (not the NULL value).

Unintentional Insertion of Zeros or Empty Strings into a Table

Execution of one of the navigation functions fetches NULL values into InTouch tags as zeros or zero-length strings (e.g. Tag1). After some other tags in the BindList are updated, execution of SQLInsert or SQLInsertExecute persists the zeros or zero-length strings (of Tag1) into the table (not the NULL value).

C H A P T E R 5

Troubleshooting

This chapter explains how to troubleshoot SQL applications using the Result Codes returned by SQL functions. The first section describes the **SQLErrorMsg()** function and includes a table of SQL Result Codes with their corresponding Error Messages. The second section includes tables with specific database Error Messages.

Contents

Troubleshooting Functions

All SQL Functions return a *Result Code* that can be used for troubleshooting. The **SQLErrorMsg()** function returns the Error Message associated with the *Result Code*.

Example

```
ErrorMsg=SQLErrorMsg(ResultCode);
```

where:

ErrorMsg is a memory message tag.

ResultCode is an integer value obtained from a previous SQL function.

Result Code Error Messages

For Result Codes that are not documented here, please refer to your specific database documentation and be sure to check the Wonderware Logger for any additional information.

The **SQLErrorMsg()** function will set the value of the InTouch message tagname *ErrorMsg*. The following is a listing of some of the possible SQL Result Codes and their corresponding error messages and descriptions:

Result Code	Error Message	Description
0	No errors occurred	The command was successful
-1	<Message from DB Provider>	<A specific error message from the DB provider>
-2	An empty statement cannot be executed	SQLExecute(ConnectionId, BindList, 0) is executed without previously calling SQLSetStatement or SQLLoadStatement with a non-empty statement.
-4	Value returned was Null	An integer or real value read from the database is null. This is only a warning sent to Wonderware Logger.
-5	No more rows to fetch	The last record in the table has been reached
-7	Invalid parameter ID	SQLSetParamI*{Type}* is called with an invalid parameter ID.
-8	Invalid parameter list	Example of an invalid parameter list: 1, 2, 3, 5 (Missing 4).
-9	Invalid type for NULL parameter	SQLSetParamNull is called with an invalid type.
-10	Changing data type of parameter is not allowed	SQLSetParam *{Type}* is called with a different type for an existing parameter.
-11	Adding parameter after the statement has been executed successfully is not allowed.	SQLSetParam *{Type}* is called for a new parameter ID after the statement has been executed successfully.
-12	Invalid date time format	An invalid date time format is encountered, for example, when executing SQLSetParamTime, SQLInsertExecute, or SQLUpdateCurrent.
-13	Invalid decimal format	An invalid decimal format is encountered, for example, when executing SQLSetParamDecimal, SQLInsertExecute, or SQLUpdateCurrent.
-14	Invalid currency format	An invalid currency format is encountered, for example, when executing SQLInsertExecute or SQLUpdateCurrent.
-15	Invalid statement type for this operation	SQLInsertEnd is called for a statement ID created by SQLPrepareStatement or SQLClearStatement is called for a statement ID created by SQLInsertPrepare.
-1001	Out of memory	There is insufficient memory to perform this function.
-1002	Invalid connection	The ConnectionId passed to the function is not valid.
-1003	No bind list found	The specified Bind List name does not exist.
-1004	No template found	The specified Table Template name does not exist.
-1005	Internal Error	An internal error occurred. Call Technical Support.

Result Code	Error Message	Description
-1006	String is null	Warning - the string read from the database is null. This is only a warning sent to Wonderware Logger.
-1007	String is truncated	Warning - the string read from the database is longer than 131 characters and is truncated on a select. The warning is sent to Wonderware Logger.
-1008	No Where clause	There is no Where clause on Delete.
-1009	Connection failed	Check Wonderware Logger for a more detailed description of the failed connect.
-1010	The database specified on the DB= portion of the connect string does not exist	The specified database does not exist.
-1011	No rows were selected	A SQLNumRows(), SQLFirst(), SQLNext(), SQLLast, or SQLPrev() command was attempted without executing a SQLSelect() or SQLExecute command first.
-1013	Unable to find file to load	SQLLoadStatement is called with a filename that cannot be found.

Specific Database Error Messages

Oracle

Check your Oracle Server documentation for specific error messages and solutions.

Microsoft SQL Server

Error Message	Solution
You cannot have more than one statement active at a time	You are trying to execute a SQL command after executing a **SQLSelect()**. Execute a **SQLEnd()** to free system resources from the **SQLSelect()** or; Use a separate ConnectionId for the second statement.
There is not enough memory available to process the command	Try rebooting the client workstation.
Invalid object name table name	The table name does not exist in the database you are using. Try DB=database name.

Check you Microsoft SQL Server documentation for specific error messages and solutions.

Debugging SQL Access

The SQLTrace=1 flag defined under the [InTouch] section of the win.ini file is useful for debugging SQL Access scripts. The new SQL Access module does not use the trace file sqltrace.txt.

A P P E N D I X A

Reserved Keywords

SQL Access and ODBC

This appendix lists the keywords that are excluded from use for the SQL Access Bind List and the Table Template, and the Open Database Connectivity (ODBC) interface.

If a reserved keyword is used as the Column Name in a Bind List or Table Template, an error message is generated in the Wonderware Logger. The type of error generated depends upon the ODBC driver being used and the location in which the keyword is found. For example, one of the most common errors made is using DATE and TIME for Column Names in a Bind List or Table Template. To avoid this error, use a slightly different name, for example, "aDATE" and "aTIME."

The reserved keywords define the Structured Query Language (SQL) used by InTouch SQL Access. The keywords are also recognized by the specific ODBC driver being used. SQL Access passes the SQL command containing one or more reserved keywords to the ODBC.DLL file. If the SQL command cannot be interpreted correctly, SQL Access generates an error message in the Wonderware Logger.

The reserved keywords for SQL Access and ODBC are listed alphabetically below:.

ABSOLUTE	BY	CONSTRAINT
ADA	CASCADE	CONSTRAINTS
ADD	CASCADED	CONTINUE
ALL	CASE	CONVERT
ALLOCATE	CAST	CORRESPONDING
ALTER	CATALOG	COUNT
AND	CHAR	CREATE
ANY	CHAR_LENGTH	CURRENT
ARE	CHARACTER	CURRENT_DATE
AS	CHARACTER_LENGTH	CURRENT_TIME
ASC	CHECK	CURRENT_TIMESTAMP
ASSERTION	CLOSE COALESCE	CURSOR
AT	COBOL	DATE
AUTHORIZATION	COLLATE	DAY
AVG	COLLATION	DEALLOCATE
BEGIN	COLUMN	DEC
BETWEEN	COMMIT	DECIMAL
BIT	CONNECT	DECLARE
BIT_LENGTH	CONNECTION	DEFERRABLE

DEFERRED
ENTF
DESC
DESCRIBE
DESCRIPTOR
DIAGNOSTICS
DICTIONARY
DISCONNECT
DISPLACEMENT
DISTINCT
DOMAIN
DOUBLE
DROP
ELSE
ENDEESC
EXCEPT
EXCEPTION
EXEC
EXECUTE
EXISTS
EXTERNAL
EXTRACT
FALSE
FETCH
FIRST
FLOAT
FOR FOREIGN
FORTRAN
FOUND
FROM FULL
GET
GLOBAL
GO
GOTO
GRANT
GROUP
HAVING
HOUR
IDENTITY
IGNORE
IMMEDIATE
IN
INCLUDE
INDEX
INDICATOR
INITIALLY
INNER
INPUT
INSENSITIVE
EINFÜGEN
INTEGER
INTERSECT
INTERVALL
INTO
IS
ISOLATION
JOIN
KEY
LANGUAGE
LAST
LEFT
LEVEL
LIKE
LOCAL
LOWER
MATCH
MAX
MIN
MINUTE
MODULE
MONTH
MUMPS
NAMES
NATIONAL
NCHAR
NEXT
NONE
NOT
NULL
NULLIF
NUMERIC
OCTET_LENGTH
OF
OFF
ON
ONLY
OPEN
OPRN
OPTION
OR
ORDER
OUTER
OUTPUT
OVERLAPS
PARTIAL
PASCAL
PLI
POSITION
PRECISION
PREPARE
PRESERVE
PRIMARY
PRIOR
PRIVILEGES
PROCEDURE
PUBLIC
RESTRICT
REVOKE
RIGHT
ROLLBACK
ROWS
SCHEMA
SCROLL
SECOND
SECTION
SELECT
SEQUENCE
SET
SIZE
SMALLINT
SOME
SQL
SQLCA
SQLCODE
SQLERROR
SQLSTATE
SQLWARNING
SUBSTRING
SUM
SYSTEM
TABLE
TEMPORARY
THEN
TIME
TIMESTAMP
TIMEZONE_HOUR
TIMEZONE_MINU
TO
TRANSACTION
TRANSLATE
TRANSLATION
TRUE
UNION
UNIQUE
UNKNOWN
UPDATE
UPPER
USAGE
USING
WERT
VALUES
VARCHAR
VARING
VIEW
WHEN
WHENEVER
WHERE
WITH
WORK
YEAR

InTouch

The following are reserved keywords for InTouch:

As
Call
Dim
Discrete
Integer
Message
Real
Return
RetVal

Index