

InTouch® HMI Supplementary Components Guide

Invensys Systems, Inc.

Revision A

Last Revision: 6/6/07



Copyright

© 2007 Invensys Systems, Inc. All Rights Reserved.

All rights reserved. No part of this documentation shall be reproduced, stored in a retrieval system, or transmitted by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of Invensys Systems, Inc. No copyright or patent liability is assumed with respect to the use of the information contained herein. Although every precaution has been taken in the preparation of this documentation, the publisher and the author assume no responsibility for errors or omissions. Neither is any liability assumed for damages resulting from the use of the information contained herein.

The information in this documentation is subject to change without notice and does not represent a commitment on the part of Invensys Systems, Inc. The software described in this documentation is furnished under a license or nondisclosure agreement. This software may be used or copied only in accordance with the terms of these agreements.

Invensys Systems, Inc.
26561 Rancho Parkway South
Lake Forest, CA 92630 U.S.A.
(949) 727-3200

<http://www.wonderware.com>

For comments or suggestions about the product documentation, send an e-mail message to productdocs@wonderware.com.

Trademarks

All terms mentioned in this documentation that are known to be trademarks or service marks have been appropriately capitalized. Invensys Systems, Inc. cannot attest to the accuracy of this information. Use of a term in this documentation should not be regarded as affecting the validity of any trademark or service mark.

Alarm Logger, ActiveFactory, ArcestrA, Avantis, DBDump, DBLoad, DT Analyst, FactoryFocus, FactoryOffice, FactorySuite, FactorySuite A², InBatch, InControl, IndustrialRAD, IndustrialSQL Server, InTouch, MaintenanceSuite, MuniSuite, QI Analyst, SCADAlarm, SCADASuite, SuiteLink, SuiteVoyager, WindowMaker, WindowViewer, Wonderware, and Wonderware Logger are trademarks of Invensys plc, its subsidiaries and affiliates. All other brands may be trademarks of their respective owners.

Contents

Welcome.....	9
Documentation Conventions.....	9
Technical Support	10
Chapter 1 Monitoring Manufacturing Quality with SPCPro Statistical Process Control.....	11
Configuring the SPCPro Database.....	12
Configuring a Local Microsoft Access Database	13
Saving Disk Space by Compacting the Database	14
Configuring a Distributed SQL Server Database.....	15
Configuring Collection and Analysis Options for a Group of Products	17
Configuring Products and Associated Process Limits	20
Collecting SPC Data Automatically Using Time or Event-Based Collection.....	23
Configuring SPC Database Users	24
SPCConnect() Function.....	25
SPCDisconnect() Function	26
Changing the Data Collection Stagger Time	27
Displaying and Manually Collecting SPC Data Using SPCPro Chart Wizards	28
Using the SPC Control Chart Wizard	28
Using the SPC Histogram Chart Wizard.....	36
Dynamically Changing the Displayed Product Using Indirect Datasets.....	38

Analyzing the Causes of Out-of-Control Conditions	39
Configuring Special Cause Codes.....	39
Visualizing the Distribution of Special Causes	
Using the Pareto Chart Wizard	40
Changing Limit Values Dynamically at Run Time	43
Working with SPCPro Alarms.....	45
Configuring SPCPro Alarms.....	45
Displaying and Acknowledging SPCPro Alarms	47
Entering Corrective Actions	49
Run Time Control Using DDE Items and	
Script Functions	51
Adding or Deleting Datasets at Run Time.....	65
SPCDatasetDlg() Function	65
Changing the Dataset Used by a Chart	66
SPCSelectDataset() Function	67
Changing the Displayed Product in a Chart	67
SPCSelectProduct() Function	68
SPCSetProductDisplayed() Function	68
Scrolling a Chart	69
SPCDisplayData() Function	70
SPCLocateScooter() Function	71
SPCMoveScooter() Function	71
Updating a Chart to the Current Time.....	72
Entering a New Sample	73
SPCSetMeasurement() Function.....	73
SPCSaveSample() Function.....	74
Checking for Deleted Samples.....	75
SPCSampleDeleted() Function	75
Changing the Currently Collected Product	76
SPCSetProductCollected() Function	77
Creating or Deleting Products During Run Time	78
NewProduct DDE Item	78
SPCCreateNewProduct() Function.....	79
SPCDeleteProduct() Function	79
Entering Calculation Parameters for Various	
Chart Types.....	80
SPCSetControlLimits() Function	80
SPCSetRangeLimits() Function	81
SPCSetSpecLimits() Function	81
SPCSetProductControlLimits() Function	82
SPCSetProductRangeLimits() Function	83
SPCSetProductSpecLimits() Function	84

Entering Data into Attribute Charts	85
Controlling Configuration Options	85
Migrating SPC Databases from a Previous SPCPro Version	86
Chapter 2 Using Recipe Manager	89
Overview of Recipe Manager	90
Recipe Manager Utility	91
Recipe Template Files	92
Template Definition	92
Unit Definition	92
Recipe Definition	92
Editing Recipe Data in Recipe Manager	93
Configuring the Recipe Manager Editing Grid	93
Working with the Editing Grid	95
Defining Ingredient Names and Data Types	99
Mapping InTouch Tags to Ingredients	100
Defining Values for Ingredients in Different Recipes	101
Editing Recipe Data in Other Applications	103
Using Excel with a Recipe Template File	103
Using Notepad with a Recipe Template File	104
Nesting Recipes to Create Complex Structures	105
Using Recipes in InTouch	106
Loading and Saving Recipe Data From/to a Recipe File	107
RecipeLoad() Function	107
RecipeSave() Function	108
Deleting Recipes From a Recipe File	109
RecipeDelete() Function	109
Selecting Units (Tag Ingredient Mappings)	110
RecipeSelectUnit() Function	110
Selecting Individual Recipes from a Recipe File	111
RecipeSelectRecipe() Function	111
RecipeSelectNextRecipe() Function	112
RecipeSelectPreviousRecipe() Function	113
Understanding Error Messages Returned by Recipe Script Functions	114
Displaying Error Code Messages	114
RecipeGetMessage() Function	116
Applying Security to Recipes	117

Chapter 3 Working with SQL Databases from InTouch	119
Setting Up a Data Source	121
Mapping InTouch Tags to Database Columns.....	122
Configuring the SQL Server String Delimiter in Bind Lists.....	124
Defining the Structure of a New Table	126
Working with Database Applications.....	128
SQL Server Database Applications	128
Microsoft Access Database Applications.....	130
Oracle Database Applications	131
Performing Common SQL Operations in InTouch	132
Connecting and Disconnecting the Database	138
SQLConnect() Function	138
SQLDisconnect() Function.....	139
Creating a New Table	139
SQLCreateTable() Function.....	139
Deleting a Table	140
SQLDropTable() Function	140
Retrieving Data from a Table	141
SQLSelect() Function	142
SQLGetRecord() Function.....	145
SQLNumRows() Function.....	146
SQLFirst() Function.....	146
SQLNext() Function.....	147
SQLPrev() Function	147
SQLLast() Function.....	148
SQLEnd() Function	148
Writing New Records to a Table.....	149
SQLInsert() Function	149
SQLInsertPrepare() Function.....	150
SQLInsertExecute() Function.....	151
SQLInsertEnd() Function	152
Updating Existing Records in a Table	153
SQLUpdate() Function.....	153
SQLUpdateCurrent() Function	155
Deleting Records from a Table	156
SQLClearTable() Function.....	156
SQLDelete() Function	157
Executing Parameterized Statements	158
SQLSetStatement() Function	158
SQLAppendStatement() Function.....	159

Creating a Statement or Loading an Existing Statement from a File.....	160
SQLLoadStatement() Function	160
Preparing a Statement.....	161
SQLPrepareStatement() Function	162
Setting Statement Parameters	163
SQLSetParamChar() Function	163
SQLSetParamDate() Function.....	164
SQLSetParamDateTime() Function	165
SQLSetParamDecimal() Function	166
SQLSetParamFloat() Function.....	167
SQLSetParamInt() Function	168
SQLSetParamLong() Function	169
SQLSetParamNull() Function	170
SQLSetParamTime() Function	172
Clearing Statement Parameters.....	173
SQLClearParam() Function.....	173
Executing a Statement.....	174
SQLExecute() Function.....	174
Releasing Occupied Resources.....	176
SQLClearStatement() Function.....	176
Working with Transaction Sets	177
SQLTransact() Function	177
SQLCommit() Function.....	178
SQLRollback() Function.....	178
Opening the ODBC Administrator Dialog Box at Run Time	180
SQLManageDSN() Function.....	180
Understanding SQL Error Messages	181
SQLErrorMsg() Function	181
Reserved Word List	185

Chapter 4 Using the 16-Pen Trend Wizard..... 189

Creating a 16-Pen Trend.....	190
Configuring Which Tags to Display on the Trend Graph	191
Configuring the Trend Time Span and Update Rate	193
Configuring the Trend Display Options.....	194
Changing the Trend Configuration at Run Time	195
Controlling a 16-Pen Trend Wizard Using Scripts.....	196
ptGetTrendType() Function	196

ptLoadTrendCfg() Function	197
ptPanCurrentPen() Function.....	198
ptPanTime() Function	199
ptPauseTrend() Function	200
ptSaveTrendCfg() Function	201
ptSetCurrentPen() Function.....	201
ptSetPen() Function	202
ptSetPenEx() Function.....	203
ptSetTimeAxis() Function.....	204
ptSetTimeAxisToCurrent() Function	205
ptSetTrend() Function.....	205
ptSetTrendType() Function	206
ptZoomCurrentPen() Function	206
ptZoomTime() Function.....	207

Chapter 5 Symbol Factory 209

Symbol Types.....	209
Picture Wizards	210
Bitmap Wizards.....	210
Texture Wizards	210
InTouch Object	211
Using Symbol Factory	211
Getting Started Quickly.....	211
Placing a Symbol FactoryWizard in a Window	212
Configuring Symbol Options.....	213
Animating a Wizard	215
Editing a Symbol	216
Breaking a Wizard for Editing	217
Sharing a Category of Symbols on a Network	217
Making a Category Read-Only	218
Viewing Category Properties	218
Editing an Existing Category	219
Deleting a Category.....	219
Configuring Symbol Factory	220
Troubleshooting	221

Index 223

Welcome

The InTouch supplementary components include:

- SPCPro, a set of statistical analysis tools to monitor and measure manufacturing quality.
- Recipe Manager, a tool for managing your manufacturing recipes.
- SQL Access Manager, a tool to access, modify, create, and delete database tables.
- 16-Pen Trend, a wizard you can use to create real-time and historical trends capable of displaying data from up to 16 tags.
- Symbol Factory, a collection of over 4,000 industrial automation wizards.

This documentation assumes you know how to use Microsoft Windows, including navigating menus, moving from application to application, and drawing objects on the screen. If you need help with these tasks, see the Microsoft documentation.

Documentation Conventions

This documentation uses the following conventions:

Convention	Used for
Initial Capitals	Paths and file names.
Bold	Menus, commands, dialog box names, and dialog box options.
Monospace	Code samples and display text.

Technical Support

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

Before you contact Technical Support, refer to the relevant section(s) in this documentation for a possible solution to the problem. If you need to contact technical support for help, have the following information ready:

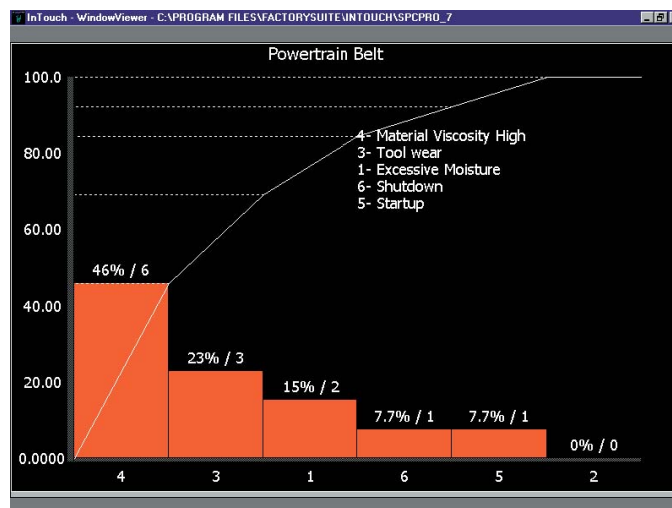
- The type and version of the operating system you are using.
- Details of how to recreate the problem.
- The exact wording of the error messages you saw.
- Any relevant output listing from the Log Viewer or any other diagnostic applications.
- Details of what you did to try to solve the problem(s) and your results.
- If known, the Wonderware Technical Support case number assigned to your problem, if this is an ongoing problem.

Chapter 1

Monitoring Manufacturing Quality with SPCPro Statistical Process Control

SPCPro™ is a Statistical Process Control (SPC) utility available with the InTouch HMI. SPC is a method for monitoring and controlling a process by gathering data about the characteristics of the output, analyzing the data, and making process management decisions based on that data.

Use the SPCPro analytical tools to monitor and measure manufacturing quality. You do not need to have a formal knowledge of statistics to use SPCPro. Graphical tools and alarm acknowledgement make SPCPro easy to use and understand.



Using SPCPro, you can:

- Monitor process quality in real time using statistical methods and alarms for rule violations.
- Get immediate feedback about your processes to quickly recognize conditions and take effective corrective actions.
- Access SPCPro data for additional post-process analysis or reporting.

Wonderware also offers the QI Analyst product, which is a more flexible statistical solution than SPCPro.

SPCPro is licensed separately from the InTouch HMI.

Configuring the SPCPro Database

Before you can use SPCPro, you must select a database to store your SPC configuration and collection data. SPCPro can be used with either Microsoft Access or Microsoft SQL Server databases.

The type of SPCPro application you want to create determines which database you can use.

SPCPro Node Type	Supported Databases
Single node	Microsoft Access or SQL Server
Multi-node	Microsoft SQL Server

You need to configure SPCPro before you can use it. You must have Microsoft ODBC drivers installed to use SPCPro. SPCPro supports the following Microsoft ODBC drivers:

- Microsoft Access driver version 4.00.3711.08 or later
- Microsoft SQL Server version 3.70.06.23 or later

Important You must set up your new SPCPro database and import any datasets created with earlier versions of SPCPro before running your InTouch application.

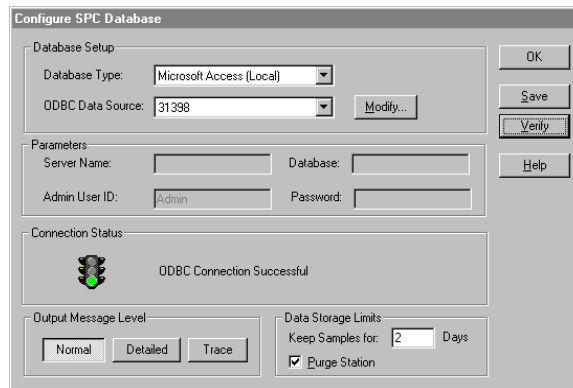
Configuring a Local Microsoft Access Database

Use the SPCPro database utility to configure an Access database for a single node application.

Note The **ODBC Microsoft Access Setup** dialog box also appears when you click the **Modify** button in the **Configure SPC Database** dialog box to edit an existing database configuration.

To configure a single node database

- 1 Start WindowMaker.
- 2 On the **Special** menu, point to **SPC**, and then click **Database**. The **Configure SPC Database** dialog box appears.



Note To close the Configure SPC Database without configuring a database, press your ESCAPE key.

- 3 In the **Database Type** list, click **Microsoft Access (Local)**.
- 4 Configure the ODBC data source. Do the following:
 - a In the **ODBC Data Source** list, click **<NEW>**. The **ODBC Data Source Administrator** dialog box appears.
 - b Click the **User DSN** tab, and then select your ODBC data source from the list or click **Add**. The **Create New Data Source** dialog box appears.
 - c Select your ODBC driver from the list, and then click **Finish**. The **ODBC Microsoft Access Setup** dialog box appears.

- d In the **Data Source Name** box, type a unique name for your data source.
 - e Click **Create**. The **New Database** dialog box appears.
 - f In the **Database Name** box, type a name for your new database.
 - g Select a folder to store your new database file and click **OK**. When a message appears, click **OK**. The **ODBC Microsoft Access Setup** dialog box reappears.
 - h Select your newly created data source.
- 5 Click **OK**. The **Configure SPC Database** dialog reappears.
 - 6 Click **Save**. When a message appears, click **Yes** to initialize the database. When a status message appears, click **OK**.
 - 7 Click **Verify**. If your ODBC connection is valid, a green light appears under **Connection Status**.
 - 8 Click **OK**.

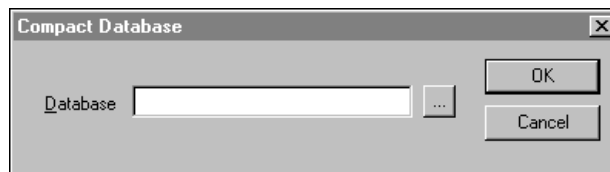
Saving Disk Space by Compacting the Database

Use the `sputil` utility to manage the size of your Access database. Access databases grow rapidly and should be compacted regularly. You see a significant reduction in the size of the database file after compacting the database.

A backup copy of the old database is saved before the compaction starts. The `sputil` utility creates a backup file and places it in the database folder. After you verify the compacted database is valid, you can delete the backup database file to free additional disk space.

To compact a Microsoft Access database

- 1 Verify that SPCPro is not running.
- 2 Run `sputil.exe`. The **sputil** dialog box appears.
- 3 On the **File** menu, click **Compact Database**. The **Compact Database** dialog appears.



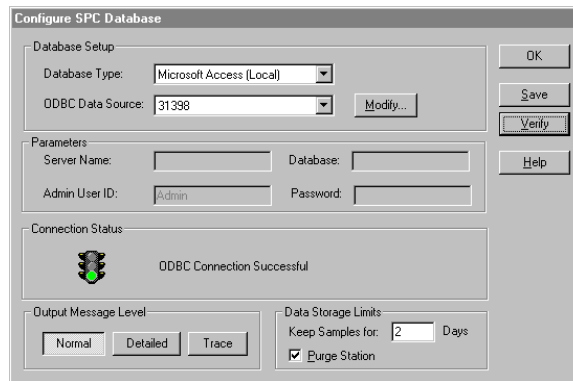
- 4 In the **Database** box, type the folder location and name of the Access database file.
- 5 Click **OK**.

Configuring a Distributed SQL Server Database

To configure an SQL database for use with SPCPro, you must first create a Microsoft SQL Server database. Also, you must have SQL Server system administrator privileges for the database to be used with SPCPro.

To configure a Microsoft SQL Server database

- 1 Start WindowMaker.
- 2 On the **Special** menu, point to **SPC**, and then click **Database**. The **Configure SPC Database** dialog box appears.



- 3 In the **Database Type** list, click **Microsoft SQL Server**.
- 4 In the **ODBC Data Source** box, select an ODBC data source by one of the following methods:
 - Select an existing ODBC data source from the list.
 - Select **<New>** from the list. The **ODBC Data Source Administrator** dialog box appears. Complete the dialog box sequence of the ODBC wizard to create a new ODBC data source.
- 5 Click **Save**.
- 6 Click **Verify** to confirm that you have set up a valid ODBC connection to the database. In the **Connection Status** area, the light should be green with a message that indicates the ODBC connection is successful.

- 7 Set the remainder of your database options.
 - a In the **Admin User ID** box, enter the name of your SQL Server administrator user account.
 - b In the **Password** box, enter the password for the administrator user account.
 - c In the **Output Message Level** area, click the option to set the level of message logging. **Normal** is the default level, which only logs error messages to the Wonderware Log Viewer. **Detailed** and **Trace** should be selected only for diagnostic purposes. Additional ODBC messages are written to the Wonderware Log Viewer. Selecting **Detailed** or **Trace** can impact system performance.
 - d In the **Data Storage Limits** area, enter the number of days in the **Keep Samples for** box that you want to retain data. Data is retained for the specified number of days plus the current day. After the number of days is exceeded, the data is purged from the database.
 - e Select the **Purge Station** check box. Only one station can be set to purge at a time. Application performance is impacted if you do not periodically purge data from the database. By default, this field is set to zero and data is never deleted. If you use the default, it is recommended that you periodically purge or archive data to avoid running out of disk space.
- 8 Click **OK** to close the dialog box.

Configuring Collection and Analysis Options for a Group of Products

You must configure datasets or indirect datasets for your SPC applications. Before configuring your SPC dataset, define these InTouch tags:

- The SPC collection tag.
- A scooter tag, if you want to use a scooter in your chart.
- A tag that increments as a counter, if you plan to use event-based collections.

Note The WindowViewer tag usage count does not include any tag used by SPCPro as a collection tag, scooter tag, or trigger tag.

SPCPro uses six-digit floating-point numbers for all values that appear in windows or assigned as values of DDE items. SPCPro chart values are rounded if they exceed six-digit precision.

To configure a SPC dataset

- 1 Start WindowMaker.
- 2 On the **Special** menu, point to **SPC**, and then click **Datasets**. The **SPC Datasets Configuration** dialog box appears.

The screenshot shows the 'SPC Dataset Configuration' dialog box with the following settings:

- Dataset Name:** ProdLine1
- Collection Tagname:** DR000
- Scooter Tagname:** Scooter1
- Analysis:** X Bar, R
- Samples Per Chart:** Control: 20, Histogram: 20, Pareto: 20
- Sample Info:** Size: 100
- Data Collection:**
 - Manual-Only
 - Time-Based: Seconds Between Measurements: 5, Minutes Between Samples: 1
 - Event-Based: Increment Tagname: [empty]
- Control Limits:**
 - Auto Calculate Every: 10 Samples
 - Samples Per Limit Calculation: 20
- EWMA Parameters:**
 - Tighter Control (2.58 sigma)
 - Smoothing Factor: 0.35

Buttons on the right side include: OK, Cancel, New, Duplicate, Save, Delete, Restore, Causes, Alarms, and Products.

- 3 Select the dataset. Do either of the following:
 - Click **New** to create a new dataset. In the **Dataset Name** box, type the name of a new dataset. The dialog box closes and the selected dataset name is automatically inserted into the **Dataset Name** box.
 - Click **Select** to choose an existing dataset. The **Select a Dataset** dialog box appears for you to select the dataset that you want to use.
- 4 Specify the tags to use. Do the following:
 - a In the **Collection Tagname** box, type the name of the collection tag or double-click the blank box and select the tag from the **Select a Tag** dialog box.
 - b Select the **Scooter Tagname** check box if you plan to use a scooter with your SPC chart, and then type the name of an analog (real or integer) memory tag in the box.
- 5 Click **Analysis**. The **SPC Analysis Selection** dialog box appears. In the **Analysis Type** area, select the analysis type that you want to use for this dataset, and then click **OK**.
- 6 In the **Samples Per Chart** area, type the number of samples to be shown for each SPC chart type or use the default of 20.
- 7 In the **Sample Info** area, in the **Size** box, type the sample size. This option is only available when you select an NP chart.
- 8 In the **Data Collection** area, configure the data collection method. Do any of the following:
 - Click **Manual Only** to have users click on the SPC chart to enter a sample. Select this method if you are using QuickScripts to log samples to the dataset. You can automate manual data collection using a timed script.
 - Click **Time Based** for SPCPro to collect data at periodic intervals. Select an agent for automatic data collection by clicking **Agent** and selecting a user from the **SPC Users** dialog box. Type values in the **Seconds Between Measurements** and **Minutes Between Samples** boxes.
 - Click **Event Based** to do event-based data collection. In the **Increment Tagname** box, type the name of a discrete InTouch tag. Select an agent for event-based collection.

- 9 In the **Control Limits** area, set control limit options. Do the following:
 - a Select the **Auto Calculate Every** check box for SPCPro to automatically calculate your control limits. In the **Samples** box, type the frequency at which the automatic control limit calculation is performed.
 - b In the **Samples Per Limit Calculation** box, type the number of samples to include in the limit calculation.
- 10 If you selected EWMA as the analysis type, select the options in the **EWMA Parameters** area.
 - a Select the **Tighter Control(2.58 sigma)** check box if you want tighter limits on your exponentially weighted moving averages.
 - b In the **Smoothing Factor** box, type the smoothing factor that you want to use. The default is 0.35.
- 11 Click **Save**. The **Products**, **Alarms**, and **Causes** buttons become available.

Note After you have configured a new dataset, you must configure at least one product for it before you can close the **SPC Database Configuration** dialog box.

Configuring Products and Associated Process Limits

You must define at least one product for an SPC dataset. You can also define other products for the same SPC dataset.

Use multiple products if you have manufacturing processes that use the same equipment to produce several different products. For example, a SPC dataset monitors the temperature of a mixer. The mixer's temperature set point and system response vary based upon the product being produced.

If you use multiple product names, you can automatically change all of the chart variables whenever you change products by using the ProductCollected DDE item.

When a change occurs in this item, the SPCPro program searches its files for the last time the product was run and uses the last chart variables as the starting point for the new data collection interval.

Note After you configure a new dataset, you must configure at least one product for it before you can close the **SPC Database Configuration** dialog box.

To configure products for datasets

- 1 Start WindowMaker.
- 2 On the **Special** menu, point to **SPC**, and then click **Datasets**. The **SPC Datasets Configuration** dialog box appears.
- 3 Click **Products**. The **Products** dialog box appears.

- 4 In the **Name** box, type the product name, with a maximum of 64 characters.

- 5 In the **Center Chart** area, set values for the chart's control limits, specification limits, center line, and target. Do the following:
 - a In the **UCL** box, type the upper control limit.
 - b In the **USL** box, type the upper specification limit.
 - c In the **Center** box, type the center value.
 - d In the **Target** box, type the chart's target value.
 - e In the **LCL** box, type the lower control limit.
 - f In the **LSL** box, type the lower specification limit.
- 6 In the **Width Chart** area, set the mean and the control limits of range or standard deviation charts. Do the following:
 - a In the **UCL** box, type the upper control limit.
 - b In the **Mean** box, type the chart mean.
 - c In the **LCL** box, type the lower control limit.

New values are used as the defaults for the next sample. You can change these limits through DDE at run time. The dataset keeps a separate copy of chart values for each configured product. This option is available for **X Bar, R, X bar, s**, and **Moving X, Moving R** charts.
- 7 In the **Sample Info** area, in the **Measurements Per Sample** box, type the number of measurements to calculate the sample point. Valid values are between 2 and 300. This option is available for **X Bar, R** or **X bar, s** charts.
- 8 In the **Display Titles** area, enter titles for each chart type for the product.
- 9 Click **Save**. To add another product, click **New** and go back to step 3 to configure the product.
- 10 When you have configured all products, click **OK**. The **SPC Dataset Configuration** dialog box reappears.
- 11 Click **OK** to close the **SPC Dataset Configuration** dialog box.

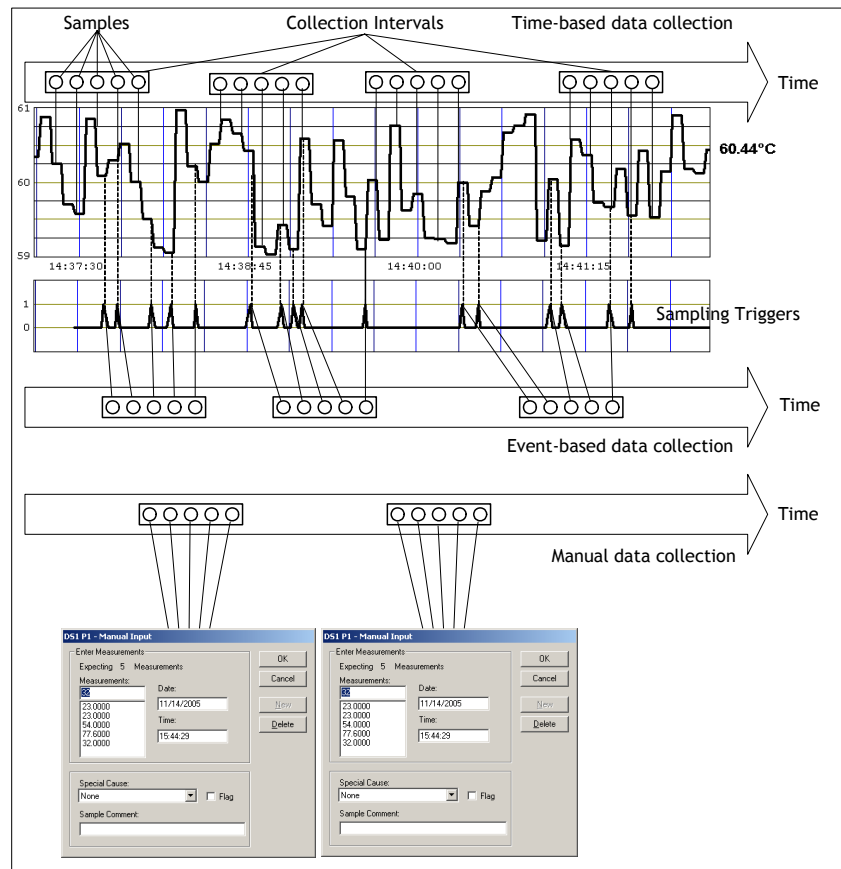
To edit an existing product

- 1 Start WindowMaker.
- 2 On the **Special** menu, point to **SPC**, and then click **Datasets**. The **SPC Datasets Configuration** dialog box appears.
- 3 Click **Products**. The **Products** dialog box appears.
- 4 Click **Select**. The **Select a Product** dialog box appears.
- 5 Select the product that you want to modify. The **Products** dialog box reappears displaying the selected product's configuration settings.
- 6 Make the required modifications and click **OK**.

Collecting SPC Data Automatically Using Time or Event-Based Collection

Typically, you collect SPCPro data to a dataset automatically using time-based or event-based sampling. Automatic data collection requires that an agent or user be connected to the database that stores statistical data.

The following figure shows how SPCPro automatically collects statistical data based on time-based and event-based sampling.



Time-based collection saves data to the dataset at a fixed interval for a specified duration. For example, a time-based dataset can be configured to collect samples over a 10 second period that occurs every 5 minutes.

Event-based data collection requires a unique discrete tag that serves as a trigger to initiate a sampling period. When the discrete tag is set to true, the dataset collects samples over a period, and then the tag resets to false.

You can also use SPCPro wizards to manually collect data. For more information about manually collecting data, see [Displaying and Manually Collecting SPC Data Using SPCPro Chart Wizards](#) on page 28.

Configuring SPC Database Users

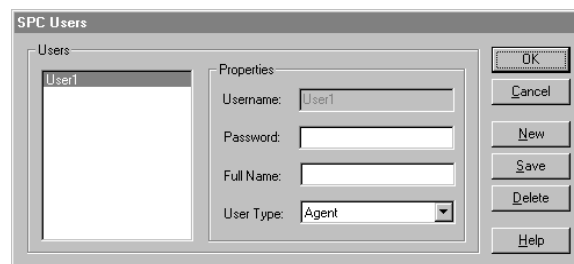
To perform automatic dataset collection, you must configure SPC database users and passwords. However, if you want to manually collect data for SPCPro, an SPC user does not need to be configured.

Note Your database must be configured prior to setting up user files.

SPC users are for SPCPro only and not SQL Server. SPCPro users are not added to the SQL Server's user list.

To configure an SPC database user

- 1 Start WindowMaker.
- 2 On the **Special** menu, point to **SPC**, and then click **Users**. The **SPC Users** dialog box appears.



- 3 Click **Add**. The options in the **Properties** area become active.
- 4 Set the properties of the new user.
 - a In the **Username** box, enter the user name.
 - b In the **Password** box, enter the user password. The password is used as an argument of the `SPCConnect()` function to connect to the database. The **Password** box can be left blank for databases that do not require user passwords.
 - c Optionally, type the full user name in the **Full Name** box.
 - d In the **User Type** list, click **Agent**.
- 5 Click **OK** or click **Save** to configure another user.

SPCConnect() Function

You connect an agent or user to a dataset with the SPCConnect() function. The SPCConnect() function identifies a user before an automatic data collection dataset starts collecting data. The SPCConnect() function must be used to indicate to SPCPro which User this node is.

You can use SPCConnect() to connect to the database when the application starts. You can use the SPCConnectType DDE message tag to monitor whether you are connected as a client or agent (Server).

Running a script that includes the SPCConnect() function connects the user to the database and starts all automatic data collection datasets using this user ID. Every dataset that agent is configured for collects data as long as that agent remains connected.

Although an agent can be associated with multiple datasets, each dataset can only have one agent assigned to it. Also, only one agent can be connected at a time from each InTouch application.

The current agent must disconnect using the SPCDisconnect() function before another agent can connect to the SPCPro engine. Also, the same agent can connect from another InTouch node.

Category

SPC

Syntax

```
SPCConnect ("User", "Password");
```

Arguments

User

Database user name. Actual string or message tag.

Password

User's password. Actual string or message tag.

Note The SPCConnect() function *User* and *Password* arguments are case-sensitive. If you do not see the message Successful connection for user XXXX in the Wonderware Log Viewer, then the agent is not connected and no data is being collected for the dataset.

Remarks

SPCConnect() connects an agent or user to a database and starts all automatic data collection datasets using this user ID. If no password is required to connect to the database, do not assign a value to the SPCConnect() *Password* argument.

Example

This example connects User1 to the SPCPro database without a password.

```
SPCConnect("User1", "");
```

See Also

SPCDisconnect()

SPCDisconnect() Function

The SPCDisconnect() function disconnects an agent from an SPCPro database. All automatic datasets assigned to the agent stop collecting data.

Category

SPC

Syntax

```
SPCDisconnect();
```

Arguments

None.

Example

This example disconnects the agent from the SPCPro database. All automatic datasets assigned to the agent stop collecting data.

```
SPCDisconnect();
```

Changing the Data Collection Stagger Time

Timed-based dataset collection periods are automatically staggered to improve system performance. This reduces system load by minimizing the period when multiple datasets are collecting data simultaneously. The default stagger period is two seconds.

You should change the dataset stagger value if you are using time-based collection at 1 minute intervals and collecting more than 30 datasets. The default stagger value is two seconds.

To change data collection stagger time

- 1 If needed, stop the application in WindowViewer.
- 2 Edit the spc.ini file in the InTouch application folder.
- 3 Add a new line to the file based upon the number of datasets collecting data.

30-40 datasets

```
StaggerValue = 1500
```

41-60 datasets

```
StaggerValue = 1000
```

Stagger values are expressed in milliseconds.

- 4 Save your changes to the spc.ini file.

Displaying and Manually Collecting SPC Data Using SPCPro Chart Wizards

You can use SPC Chart wizards to create various types of statistical charts. SPCPro provides wizards to create three types of SPC charts:

- Control Charts
- Histograms
- Pareto charts.

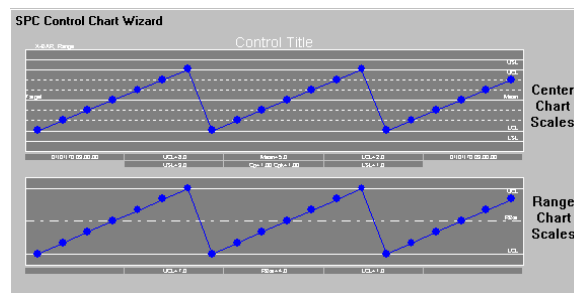
SPC chart objects are wizards that you paste into an InTouch window. Then, you configure the SPC charts with the wizards and link the charts to the datasets containing statistical data.

You can configure the SPC Control Chart wizard to display the following chart types:

X Individual	X bar - R chart	X bar - s chart
Cumulative Sum (CUSUM) chart	Moving-X and Moving-R charts	EWMA chart
C chart	P chart	U chart
NP Chart		

Using the SPC Control Chart Wizard

The Control Chart wizard creates X-Y charts that show a calculated point for each sample or sub-group. The points are connected by lines to form a control chart, which provides a graphical representation of the values associated with a monitored process.



A center line is used to show the average of all points in a group. An upper control limit line is shown at three standard deviations above the center line. The lower control limit line is three standard deviations beneath the center line.

Upper and lower specification limit lines show arbitrary top and bottom limits of acceptable output. A target line shows the desired average value of the process, which should be the same as the center line. Zone lines are reference lines that are plus and minus one and two standard deviations away from the center line.

If a sample falls outside of the control limits (or breaks one of the run rules), an alarm occurs to notify the user about a special cause that produced the out of control sample.

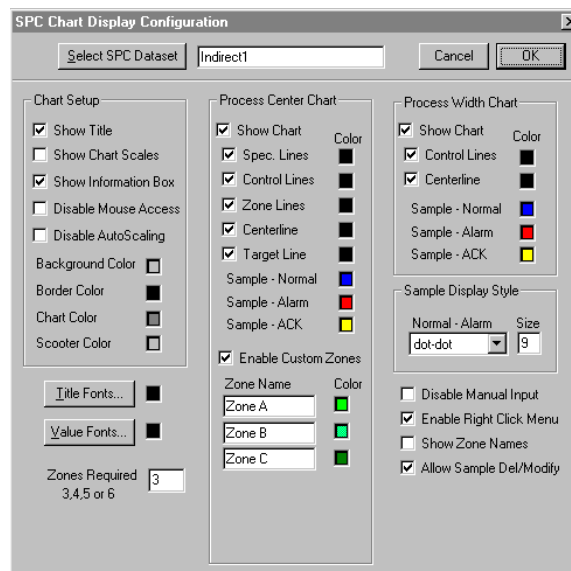
SPCPro provides three SPC chart wizards and a SPC limits wizard. To use the wizards, you must first place them in an open window from WindowMaker. Then, configure the properties and link the chart wizard to a dataset.

For more information on placing wizards in a window, see Chapter 5, Wizards, in the *InTouch® HMI Visualization Guide*. The wizard category is **SPC Charts**.

When a SPC Control Chart wizard is used, it must be configured to link the SPC Control Chart to your SPC dataset.

To configure a Control Chart

- 1 If needed, start WindowMaker.
- 2 Open the window that contains an SPC Control Chart wizard.
- 3 Double-click on the SPC Control Chart wizard. The **SPC Chart Display Configuration** dialog box appears.



- 4 Enter the name of the dataset to be used with the SPC Control chart wizard by one of the following methods:
 - Type the dataset name in the input box near the top center of the dialog box.
 - Click **Select SPC Dataset**. The **Select a Dataset** dialog box appears. Select a dataset name from the list and click **OK**.

- 5 In the **Chart Setup** area, set the options that determine the visual appearance of the chart:
 - To place a title bar above the chart, select the **Show Title** check box.
 - To show scales adjacent to the chart, select the **Show Chart Scales** check box.
 - To show a chart information box, select the **Show Information Box** check box.
 - To disable chart touch-sensitivity in run time, select the **Disable Mouse Access** check box.

If this option is not selected and you click a sample character in the chart during run time, the **Sample Information** dialog box appears.

- To select the way the chart calculates the display range, select the **Disable Autoscaling** check box.

Normally, the chart display range is calculated to accommodate all displayed samples, control limits, and specification limits. If this option is selected, the chart range is determined as follows:

If the chart is configured to display control limits and specification limits, the range expands to show all displayed control and specification limit values.

If the chart is configured to display control limits and not specification limits, the range adjusts to show all control limit values.

If neither control limits nor specification limits are configured, the range of the chart is determined by the current specification limit values.

- 6 To set chart colors, do the following:
 - a Click the color box next to each color option to show a color palette.
 - b Select the desired color from the palette.

- 7 Select the fonts used by the chart:
 - a Click **Title Fonts** to open the **Font** dialog box. Select the font, font style and size for the chart's title. Click the color box to select the color in the color palette that you want to use for the title.
 - b Click **Value Fonts** to open the **Font** dialog box. Select the font, font style and size for the values shown on the chart. Click the color box to select a color from the palette that you want to use for the values.
- 8 In the **Zones Required 3,4,5 or 6** box, enter the number of zones that appear in your chart.
- 9 In the **Process Center Chart** area, set the options that determine the lines shown in a center chart:
 - a Select the **Show Chart** option to configure the lines you want shown on your Center Chart.
 - b Select the check box next to the chart lines you want to appear in the chart.
 - c Click the color box to select the color from the palette that you want to use for the **Spec. Lines, Control Lines, Zone Lines, Centerline** and **Target Line**.
 - d Set the colors for the **Normal, Alarm** and **ACK** sample points.
 - e Select **Enable Custom Zones** if you want to show levels of zones within your Center Chart.
- 10 In the **Process Width Chart** area, set the options to define and display a Width Chart. You must select the **Show Chart** option to configure the lines you want shown on your Width Chart.
- 11 In the **Sample Display Style** area, you can change the sample style and the point character size used in both your Center Chart and your Width Chart.

12 Set the options that determine what the user can do with the chart during run time:

- To prevent the user from manually entering sample values, select **Disable Manual Input**.
- To enable a right-click action menu with user options, select **Enable Right Click Menu**.

This menu includes **Acknowledging Alarms, Deleting samples, Modifying samples, Zone Centering, and Adding and Deleting Special Causes** options.

- To show zone names on a Center chart, select **Show Zone Names**.
- To permit the user to delete and modify sample values through the right-click menu, select **Allow Sample Del/Modify**.

13 Click **OK** to save your configuration values.

Detailed sample information can be obtained for any sample point. Comments and special causes can also be associated with any sample. Sample information is available either through DDE or through the **Sample Information** dialog box that appears in run time when the operator clicks a sample.

To access the Sample Information dialog

1 In run time, click the sample currently shown in the SPC Control Chart. The **Sample Information** dialog box appears.

Dataset1 Product1 - Sample Information

Sample No. 2835 << >> Sample (X) 39.1785030364 OK
 Date Time: 09/16/97 16:07:01 Sample (R) 25.1484737396 Cancel
 New

X-Chart **R,S Chart**
 UCL: 60 USL: 62 UCL: 62.6932269804
 Mean: 50.2077091598 Target: 50 Mean: 24.9286787986
 LCL: 40 LSL: 38 LCL: 0

Alarms: 4 of the last 5 samples are outside 1 sd (same side).
 X-Bar outside control limits. **Corrective Action**

Measurements:
 31.5784
 36.7283
 35.2651
 33.9735
 38.3486 **Modify**

Comment:
 Note Text:
 Special Causes: None Options:
 Flag Sample
 Ignore Value

- The **Sample No.** box shows the number of the sample selected from the SPC Control Chart.
- The **Date Time** box shows the date and time the sample is collected.

- Chart values for the Analysis Type selected are shown.
 - The **Alarms** window shows all alarm conditions for the displayed sample.
 - The **Measurements** list shows the actual values of all the measurements used in the calculation of the sample.
- 2 In the **Comment** box, type a comment up to 50 characters about the sample.
 - 3 In the **Note Text** box, type a note up to 12 characters that appears on the chart.
 - 4 In the **Special Causes** box, type a special cause for the sample.
 - 5 Select **Flag Sample** if you want to flag the sample on the chart.
 - 6 Select **Ignore Value** to force the SPC Control Chart to be redrawn, ignoring the selected sample from the auto-scaling calculation.

The sample is still plotted, but appears to be off the chart display. The sample value is also ignored in the Histogram plot.

Note Selecting this option does not exclude the point during control limit calculation, just in the SPC Control Chart display.

- 7 Click **New** to access the **Manual Input** dialog box to manually enter measurements for the sample.
- 8 Click **Corrective Action** to access the **Corrective Action** dialog box to take corrective action on the sample.
- 9 Click **OK**.

New samples can be added to the dataset in run time either through DDE by setting the **ManualInputDialog** DDE Item Name to 1, or by accessing the **Manual Input** dialog box.

To add samples with the Manual Input dialog box

- 1 Click a sample on a SPC Control Chart. The **Sample Information** dialog box appears.
- 2 Click **New**. The **Manual Input** dialog box appears.

- 3 In the **Measurements** box, enter the measurements for the sample:
 - a Enter a measurement value taken for the sample.
 - b Press **Enter**. The value is entered into the box below the input box.
 - c Repeat steps a and b until you enter the number of required measurements specified above the **Measurements** box.
- 4 To change the date or time, type the new information into the respective input boxes.
By default, the **Date** and **Time** boxes, show the current date and time associated with this new sample.
- 5 In the **Special Cause** box, select a special cause to the sample. Otherwise, accept the default of **None**.
- 6 Select **Flag** to flag the sample on the SPC Control Chart.
- 7 In the **Sample Comment** box, type a comment about the sample up to a maximum of 50 characters.
- 8 Click **OK** to add the measurements to the dataset and close the dialog box. The **Sample Information** dialog box reappears.
- 9 Click **OK**.

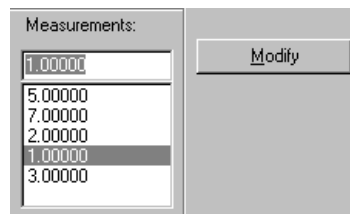
Manual input windows can be created by using the appropriate DDE items. Through DDE, manual inputs can easily be automated using an InTouch data change or condition script.

Control Chart sample information can easily be modified and deleted. However this option is not available for the CuSum, EWMA, and Moving X, Moving R dataset types.

This option must be enabled in the control chart configuration. It can be toggled on/off at run time using the SPCAllowSampleDelMod DDE Item in a script.

To modify a sample

- 1 Right-click on the point that you want to modify. An action menu appears.
- 2 Select **Modify Sample** from the menu. The **Sample Information** dialog box appears.

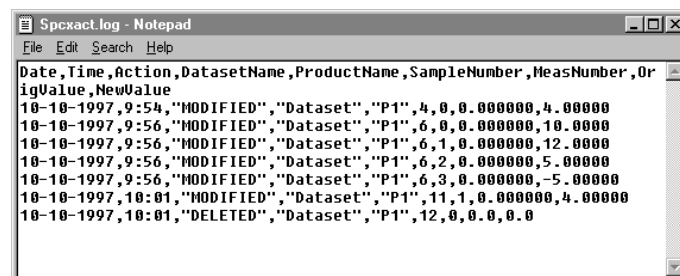


- 3 Highlight the measurement that you want to change.
- 4 Enter the new measurement and click **Modify**. The selected measurement shows the modified value.
- 5 Click **OK**.

To delete a sample

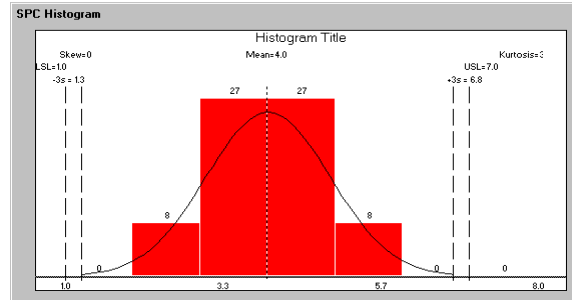
- ◆ Right-click the point that you want to delete and then click **Delete Sample**. When a message appears, click **Yes**.

Each time you delete or modify a sample, SPCPro logs the changes to a file. This log is located in the InTouch application folder as the spcxact.log file. The file shows a chronological listing of all modifications to samples.



Using the SPC Histogram Chart Wizard

Histograms are created from the raw measurement data used for the control charts. A histogram shows the distribution and frequency of collected data. A normal process distributes values in a bell-shaped curve.

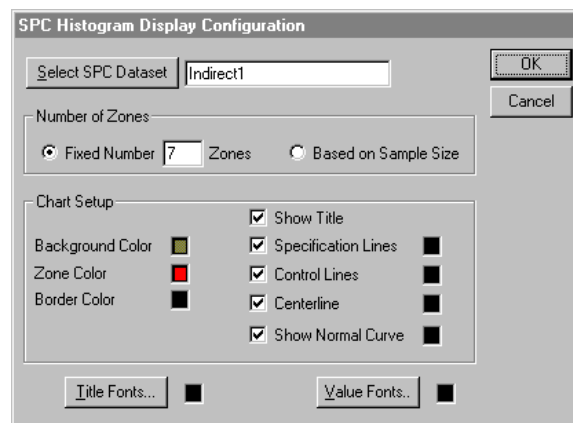


When you create a chart with the SPC Histogram wizard, you paste the wizard into a WindowMaker window and then configure and link the wizard to a dataset.

Note A previously defined dataset name must be entered to configure the SPC Histogram.

To configure a SPC Histogram wizard

- 1 Paste the SPC Histogram wizard into your window and then double-click it. The **SPC Histogram Display Configuration** dialog box appears.



- 2 Specify the name of the dataset to use with the histogram by one of the following methods:
 - Type the dataset name in the input box.
 - Click **Select SPC Dataset** and select the dataset from the **Select a Dataset** dialog box that appears.

- 3 In the **Number of Zones** area, select one of the following options to set the number of zones shown in the histogram:
 - **Fixed Number**
Creates a fixed number of zones to limit your sample size based upon the number you enter in the **Zones** box.
 - **Based on Sample Size**
Creates the zones shown on the Histogram based on your sample size.
- 4 In the **Chart Setup** area, set the options that determine the visual appearance of the histogram as follows:
 - To show a title for the histogram, select **Show Title**.
 - To show a normal distribution curve line to the histogram chart, select **Show Normal Curve**.
 - To show specification lines in the histogram, select **Specification Lines**.
 - To show control lines in the histogram, select **Control Lines**.
 - To show a center line in the histogram, select **Centerline**.
- 5 Set the colors of the histogram by doing the following:
 - a Click the color box next to each option to show the color palette.
 - b Select a color from the palette.
- 6 Select the appearance of text shown in the histogram by doing the following:
 - a Click **Title Fonts** to open the **Font** dialog box.
 - b Select the font, font style, and size of the chart's title.
 - c Click the color box to select the color in the color palette that you want to use for the Title.
 - d Click **Value Fonts** to open the **Font** dialog box.
 - e Select the font, font style, and size for the values shown on the chart.
 - f Click the color box to select the color in the color palette that you want to use for the values.
- 7 Click **OK** to save your configuration changes.

Dynamically Changing the Displayed Product Using Indirect Datasets

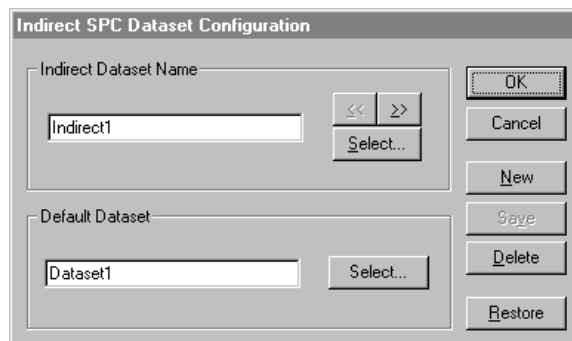
Indirect datasets allow SPC charts to be dynamically linked in run time to any dataset. With indirect datasets you can use the same SPC chart to show data from multiple datasets. To change the link at run time, you can set the DatasetName DDE item to the desired dataset.

When you configure an SPC chart, you must link it to an SPC dataset. If you link the SPC chart to an Indirect dataset, that chart can show data from any dataset. Linking a chart to an indirect dataset is accomplished by changing the DatasetName DDE item during run time. When the DatasetName DDE item is changed, the Indirect dataset assumes all of the properties and item values of the dataset to which it is linked.

Note A previously defined dataset name must be entered to configure an indirect dataset.

To configure an indirect dataset

- 1 Start WindowMaker.
- 2 On the **Special** menu, point to **SPC**, and then click **Indirect Datasets**. The **Indirect SPC Dataset Configuration** dialog box appears.



- 3 In the **Indirect Dataset Name** box, type a unique name up to 31 characters for the indirect dataset.
- 4 In the **Default Dataset** box, enter the name of the dataset to which you want to link the indirect dataset by one of the following methods:
 - Type the indirect dataset name in the **Default Dataset** box.
 - Click **Select** to show the **Select a Dataset** dialog box. All currently defined datasets are listed. Select the dataset name from the list.
- 5 Click **OK**.

Analyzing the Causes of Out-of-Control Conditions

Statistical quality control assigns deviations from acceptable standards to either common causes or special causes. Common causes are problems inherent in the manufacturing process, such as inadequate product design or excessive humidity. Special causes are local, sporadic problems such as the failure of a particular machine or a mistakenly recorded measurement. After special causes have been identified and eliminated, a process is considered to be in statistical control.

This section explains how to use SPCPro to identify special causes in your manufacturing process. The section includes information that explains how to:

- Define special causes
- Create a Pareto chart

Configuring Special Cause Codes

SPC samples that are out of control may be attributable to special causes. You can define special causes in WindowMaker through the **Special Cause Configuration** dialog box. You can attach a special cause to any sample in WindowViewer, either through the DDE item (for example **CurrentCauseCode**) or through the **Sample Information** dialog box.

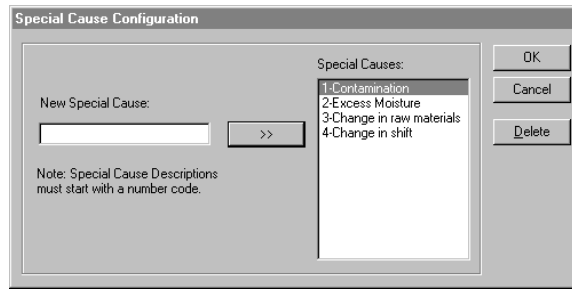
Note The **Special Cause Configuration** dialog box can be shown in run time by right-clicking a sample, and then selecting **Add/Delete Causes**.

A summary of these causes can then be shown on a Pareto chart to determine the causes that are largely responsible for out of control samples.

To configure special causes for a dataset

- 1 Start WindowMaker.
- 2 On the **Special** menu, point to **SPC**, and then click **Datasets**. The **SPC Datasets Configuration** dialog box appears.

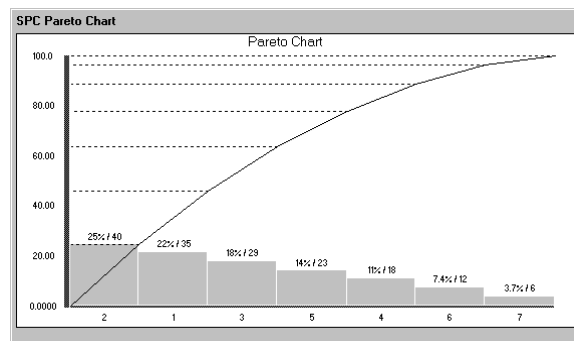
- 3 In the **SPC Dataset Configuration** dialog box, click **Causes**. The **Special Cause Configuration** dialog box appears.



- 4 Enter your cause codes by doing the following:
 - a Type a cause description with a code number in the **New Special Cause** box. This number identifies the columns in the Pareto display chart. For example, type 1-Startup
 - b Click >> button or press **ENTER** to add the new special cause to the **Special Causes** list box.
 - c Add as many special cause descriptions as you need.
- 5 Click **OK**.

Visualizing the Distribution of Special Causes Using the Pareto Chart Wizard

A Pareto Chart is a special form of a bar graph that shows the relative importance of problems or conditions. Pareto charts graphically present the number of occurrences of special causes. Since the user usually enters special cause notations when acknowledging alarms, the Pareto chart uses these entries over some specified number of samples and presents them in the form of a descending bar graph.



A Pareto chart orders data from the most frequent to the least frequent, allowing you to determine the most important factor in a given process. While there are many possible causes for out of control samples, it is usually only one or two special causes that produce the majority of bad samples.

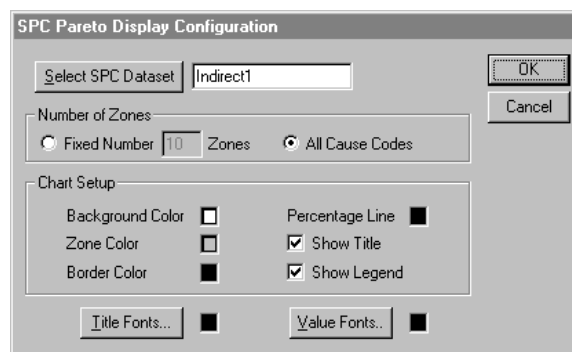
Note Pareto charts recalculate control limits for every new sample entered.

When the SPC Pareto wizard is used, it must be configured with the information required to link the SPC Pareto chart to your SPC dataset.

Note A previously defined dataset name must be entered to configure the SPC Pareto chart.

To configure a Pareto Chart

- 1 Paste the SPC Pareto wizard into your window, and then double-click it. The **SPC Pareto Display Configuration** dialog box appears.



- 2 Specify the name of the dataset to use with the Pareto chart by one of the following methods:
 - Type the dataset name in the input box.
 - Click **Select SPC Dataset** and select the dataset from the **Select a Dataset** dialog box that appears.
- 3 In the **Number of Zones** area, select one of the following options to set the number of zones shown in the Pareto chart:
 - **Fixed Number**
Creates a fixed number of zones to limit your sample size based upon the number you enter in the **Zones** box.
 - **All Cause Codes**
Select if you want the Number of Zones shown on the Pareto to be equal to the number of special cause codes.

- 4 In the **Chart Setup** area, set the options that determine the visual appearance of the Pareto chart as follows:
 - To show a title for the Pareto chart, select **Show Title**.
 - To show a legend for the Pareto chart, select **Show Legend**.
- 5 Set the colors of the Pareto chart by doing the following:
 - a Click the color box next to each option to show the color palette.
 - b Select a color from the palette.
- 6 Select the appearance of text shown in the Pareto chart by doing the following:
 - a Click **Title Fonts**, to open the **Font** dialog box.
 - b Select the font, font style, and size of the chart's title.
 - c Click the color box to select the color in the color palette that you want to use for the Title.
 - d Click **Value Fonts** to open the **Font** dialog box.
 - e Select the font, font style, and size for the values shown on the chart.
 - f Click the color box to select the color in the color palette that you want to use for the values.
- 7 Click **OK** to save your configuration changes.

Changing Limit Values Dynamically at Run Time

The SPC Limits wizard is a SPC control panel wizard that allows you to update and view SPC specification limits and SPC control limits. It also allows you to switch datasets, switch products, and scroll the SPC Chart.

XUSL #.####	Update
XUCL #.####	
Target #.####	Current X Sample #.####
MEAN #.####	Current R Sample #.####
XLCL #.####	◀ ▶ Scroll Value #
XLSL #.####	
RUCL #.####	Dataset #
RBAR #.####	
RLCL #.####	
Product	
Displayed #	
Collected #	

Note Pareto charts recalculate control limits for every new sample entered.

When the SPC Limit wizard is used, it must be configured with the information required to link it to your SPC dataset.

Note A previously defined dataset name must be entered to configure the SPC Limit wizard.

To configure the SPC Limit wizard

- 1 Paste the SPC Limit wizard into your window and then double-click it. The **Select** dialog box appears.

Dataset		
dataset2		
Dataset	Suggest	OK Cancel
Tags		
CurrentXUSL	CurrentXLSL	CurrentTarget
CurrentXUCL	CurrentXLCL	CurrentSampleBar
CurrentRUCL	CurrentRLCL	CurrentRBar
CurrentSample	CurrentR	CurrentUpdate
DatasetName	ProductCollected	ProductDisplayed
LastSampleDisplayed	Scroll	

- 2 In the **Dataset** area, specify the name of the dataset by one of the following methods:
 - Type the dataset name in the input box.
 - Click **Dataset** and select the dataset from the **Select a Dataset** dialog box that appears.
- 3 In the **Tags** area, specify the tag names that you want to associate with chart objects by doing one of the following:
 - Enter the tag names that you have defined in your Tagname Dictionary for the various items.
 - Click **Suggest** for the wizard to automatically suggest tags for each item.
 - Double-click in a blank **Tags** input box to show the **Select Tag** dialog box with a list of defined tags for the selected tag source. Select the tag from the list that you want to use and click **OK**. The tag browser closes and the selected tag is automatically inserted in the selected field.
- 4 Click **OK** to save.

Working with SPCPro Alarms

You can set up SPCPro to use data from InTouch tags to determine whether a process is in statistical control. This section explains how to set up SPCPro to:

- Identify an alarm condition
- Acknowledge alarms
- Enter corrective actions in response to alarms

Configuring SPCPro Alarms

You can use SPCPro to analyze chart data for alarm conditions. It checks for out of limit conditions and seven different Western Electric run rules. An alarm is recorded to the InTouch database and the tag specified in the dataset configuration is notified.

The specific SPC alarm can be viewed in the **Sample Information** dialog box or through SPC Alarm message DDE items.

All alarms have an assigned priority level. A valid priority level is between 1 and 999. This value represents the severity of the alarm with 1 being the most severe. By assigning priority levels to alarm ranges, you can filter critical alarms based upon their priority level.

To configure alarm conditions

- 1 Start WindowMaker.
- 2 On the **Special** menu, point to **SPC**, and then click **Datasets**. The **SPC Datasets Configuration** dialog box appears.

- 3 In the **SPC Dataset Configuration** dialog box, click **Alarms**. The **SPC Alarms Selection** dialog box appears.

Section	Alarm Description	Priority
Limit Alarms	<input checked="" type="checkbox"/> Sample Outside of Specification Limits	999
	<input checked="" type="checkbox"/> Sample Outside of Control Limits	999
Standard Deviation Alarms	<input checked="" type="checkbox"/> 2 of Last 3 Samples Outside of 2 Standard Deviations (same side)	999
	<input checked="" type="checkbox"/> 4 of Last 5 Samples Outside of 1 Standard Deviation (same side)	999
	<input type="checkbox"/> 0 of Last 0 Samples Outside of 0 Standard Deviations	999
	<input type="checkbox"/> 0 of Last 0 Samples Outside of 0 Standard Deviations (same side)	999
Consecutive Alarms	<input type="checkbox"/> Consecutive Samples Inside of 1 Standard Deviation	999
	<input type="checkbox"/> Consecutive Samples Outside of 1 Standard Deviation	999
	<input type="checkbox"/> Consecutive Samples Increasing or Decreasing	999
	<input type="checkbox"/> Consecutive Samples Alternating Up and Down	999
	<input type="checkbox"/> Consecutive Samples on One Side of the Centerline	999

- 4 In the **Limit Alarms** area, select the options to restrict alarms based upon the sample value:
- To set alarms for sample values outside of the specification limits, select **Sample Outside of Specification Limits**.
 - To set alarms for sample values outside of the control limits, select **Sample Outside of Control Limits**.
- 5 In the **Standard Deviation Alarms** area, select the options to set alarms based upon sample values outside of a chart's standard deviation limits.
- To set an alarm if 2 of the last 3 samples are outside of 2 standard deviations, select the option within the **Standard Deviation Alarms** area.
 - To set an alarm if 4 of the last 5 samples are outside of the first standard deviation, select the option within the **Standard Deviation Alarms** area.
 - If you want, create custom standard deviation alarm limits by entering the number of samples outside of standard deviation limits for the last two options listed in the **Standard Deviation Alarms** area.

The first option can include samples points from both sides of the chart's center line. The second alarm option is the same, except all samples must be from either above or below the center line. For both options, the first two fields are integers and the last field is a real number.

- 6 In the **Consecutive Alarms** area, select the options to set alarms based upon consecutive sample values outside of a chart limit.
 - a Select the check box next to the alarm option that you want to use to select alarms.
 - b Enter the number of consecutive samples for each option that sets the threshold for an alarm.
- 7 Assign a **Priority** level to each alarm condition that you selected from the **SPC Alarm Selection** dialog box.
- 8 Click **OK**.

Displaying and Acknowledging SPCPro Alarms

SPCPro provides alarm data to the InTouch Alarm manager. You can show current SPCPro alarms in the AlarmViewer ActiveX control. Alarms can be acknowledged by a right-click on the alarm sample in the SPCPro chart and selecting **Ack Sample** from the action menu. The alarm is acknowledged in the SPC chart and the AlarmViewer. Also, alarms can be acknowledged directly from the AlarmViewer ActiveX control, which then updates the SPC sample chart.

Time	State	Class	Type	Priority	Name	Group
01/15/2007 08:42:36 AM	UNACK	VALUE	HI	1	ReactLevel	React
01/15/2007 08:42:13 AM	UNACK_RTN	VALUE	HI	1	ReactTemp	React
01/15/2007 08:38:40 AM	UNACK_RTN	VALUE	HI	1	ProdLevel	React

Displaying 1 to 3 of 3 alarms. Default Query 100 % Complete

When SPCPro initializes in run time, datasets are analyzed up to the last control limit calculation to ensure correct analysis of alarms (Run Rules). For large datasets this can take a long period of time. To disable checking datasets for alarms during initialization, include the following line in the [General] section of your SPC.INI file:

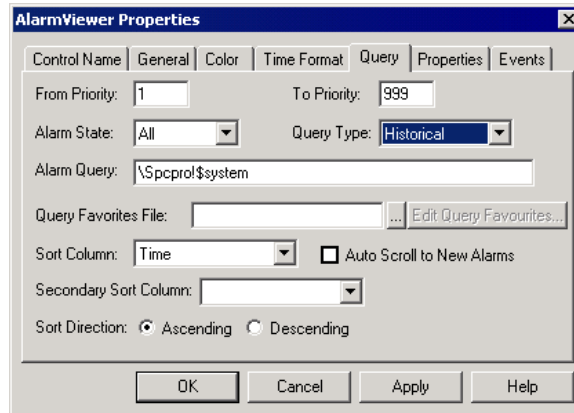
```
AlarmAnalysisOnStartup=1
```

Note SPCPro is an InTouch alarm provider and must be configured as an alarm provider in the Alarm Object.

The following procedure describes the essential steps to configure an AlarmViewer control to show SPCPro alarms. For more information about configuring all AlarmViewer properties, see *Configuring an Alarm Tree Viewer Control* in Chapter 7, *Viewing Alarm Hierarchies*, in the *InTouch® HMI Alarms and Events Guide*.

To configure the Alarm Viewer control for SPCPro alarms

- 1 Open a window containing an Alarm Viewer control.
- 2 Double-click on the Alarm Viewer control. The **AlarmViewerCtrl Properties** dialog box appears.
- 3 Click the **Query** tab to show the alarm properties of the control.



- 4 In the **Alarm Query** box, enter a node address based upon the alarm providers for SPCPro.
 - To display alarms from the local SPCPro node, enter the following:
`\SPCPro!$System`
 - To display alarms from all providers to the SPCPro server node, enter the following:
`\\NodeName\spcpro!$System`
 - To display alarms from all providers and the local SPCPro node, enter the following with a blank as a string separator:
`\Spcpro!$System \\NodeName\spcpro!$System`
- 5 Click **Apply**.
- 6 Set other Alarm Viewer control properties.
- 7 Click **OK** after setting all other properties of the Alarm Viewer control.

Entering Corrective Actions

Operators make a corrective action or control move to indicate to SPCPro that a change or modification has been made to a manufacturing process. A corrective action also allows an operator to document changes in a log that is attached to the sample number.

For example, an operator sees an alarmed point on the Control Chart. The operator observes that a control valve is opening only partially instead of fully. The operator goes out to the production line and confirms the valve is not functioning properly. Therefore, the operator decides to replace the valve. After completing the repair, the operator enters a corrective action on the sample.

The sample is identified with the symbol [c] in the SPC Control Chart to indicate that corrective action has been taken. Also, SPCPro alarm counters are reset according to the switch setting in the SPC.INI file located in the InTouch application folder. Taking a Corrective Action on the last sample (CurrentSampleNumber) also resets the SPCPro Run Rule Counts.

When a corrective action is taken, the SPC Alarm counters automatically reset to zero. This reset is controlled through a switch setting in the SPC.INI file. The default setting is to reset only those alarms that exist for the sample for which the correction action was performed. However, you can reset all SPC alarm counters by including the following line in your SPC.INI file:

```
[General]
ResetAllAlarmCounters=1
```

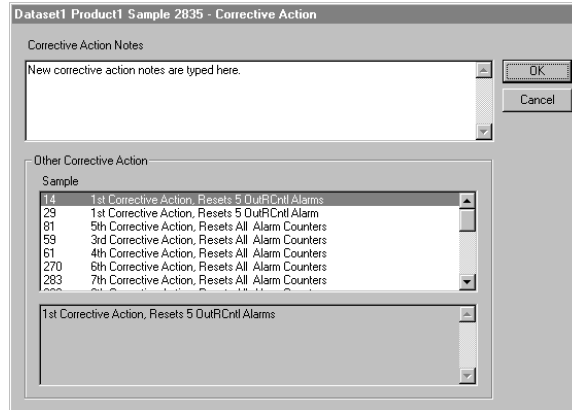
The corrective action name is controlled through a switch setting in the SPC.INI file. By default the name, Corrective Action is used. However, you can change the name to Control Move by including the following line in the SPC.INI file:

```
[General]
CorrectiveAction=0
```

This switch setting changes the name to Control Move.

To enter a corrective action

- 1 Click the alarmed sample in the SPC Control Chart. The **Sample Information** dialog box appears.
- 2 Click **Corrective Action**. The **Corrective Action** dialog box appears.



- 3 In the **Corrective Action Notes** window, type the notes regarding the corrective action taken for the sample.

Note The **Other Corrective Action** window lists all corrective actions taken for samples in the SPC dataset linked to the SPC Control Chart. If you select a listed correction action, its respective notes are shown in the lower window.

You can scroll through the text. You can also select the text, press the Ctrl+C keys to copy it, and then press the Ctrl+V keys to paste the text in the **Correction Action Notes** window. You can then modify the text as required for your corrective action.

- 4 Click **OK**. A message appears requesting confirmation to to perform the corrective action for the sample.
- 5 Click **Yes** to insert the corrective action into the SPC database. The **Corrective Action** dialog box reappears.
- 6 Click **OK**. The **Sample Information** dialog box reappears.
- 7 Click **OK**.

Run Time Control Using DDE Items and Script Functions

SPC charts can be controlled at run time by a variety of methods. You can create SPC applications that are controlled either by mouse clicks or by keyboard entries, which invoke a script or SPC DDE items.

SPC functions and DDE give you extensive control over chart objects and SPC data. This section describes the SPC DDE Items and SPC functions that you can use to control your SPC applications during run time.

DDE items can be used to obtain dataset information and control chart operations. The application name is SPCPro. The topic name is the dataset name. The topic name is case-sensitive.

The Control and Display DDE Items are used to control and display information about the topic dataset. Control DDE items are shared by all nodes. They are the dataset values for the collected product of the remote dataset. Display DDE items are local to each node. They are the sample values for the displayed product on the local node.

Sample modifications can be applied to collected and displayed products of local datasets. Modification made by clicking on a chart display affect the displayed product. The SPC DDE items modify the collected product. The displayed product and the collected product have their own current sample, which is the most recently recorded sample.

Alarms are evaluated and stored for collected and displayed products. Alarms are only reported during run time for the collected product.

With the addition of displayed and collected products, many SPC DDE Items apply only to the collected product. These items are flagged in the following table by an asterisk (*) preceding the SPC DDE Item name.

Item Name	DDE Type	Access	Description
AutoCollection	Discrete	R/W	Enables/disables automatic data collection.
*CalculateControlLimits	Discrete	R/W	Set to 1 to start control limit calculation.
DatasetName	Message(32)	R/W	Sets the dataset Name used by an Indirect dataset.

Item Name	DDE Type	Access	Description
HistogramLCL	Real	RO	Displays the Histogram's Lower control limits based on population.
HistogramMean	Real	RO	Shows the mean value of the histogram.
HistogramUCL	Real	RO	Displays the Histogram's Upper control limits based on population.
Kurtosis	Real	RO	Distribution shape of Histograms.
LastSampleDisplayed	Integer	R/W	Sets the last sample number displayed by the dataset.
*ManualInputDialog	Discrete	R/W	Set to 1 to display built-in Manual Input Dialog Box.
MeasurementsPerSample	Integer	RO	Displays the configured number of measurements per sample.
NewProduct	Message(64)	R/W	Used to create new Product Name.
NewProductCtrlTitle	Message(32)	R/W	Used to set the Control chart title of a new product created with the NewProduct Function.
NewProductParetoTitle	Message(32)	R/W	Used to set the Pareto chart title of a new product created with the NewProduct Function.
NewProductHistTitle	Message(32)	R/W	Used to set the Histogram chart title of a new product created with the NewProduct Function.
*ProductCollected	Message(32)	R/W	Changes the Product Name collected by the dataset.
ProductDisplayed	Message(32)	R/W	Changes the Product Name displayed by the dataset.
SampleSize	Integer	RO	Sample size for NP dataset.
SamplesControlChart	Integer	R/W	Sets the number of samples displayed in a Control Chart.
SamplesPerHistogram	Integer	R/W	Sets the number of samples displayed in a Histogram.
SamplesPerLimitCalc	Integer	R/W	Sets the number of samples used in a control limit calculation.

Item Name	DDE Type	Access	Description
SamplesPerPareto	Integer	R/W	Sets the number of samples used in a Pareto Chart display.
SeISPCOutSpecMsg	Message	RO	Alarm message tag for "Sample outside specification Limit."
Skewness	Real	RO	Displays the variance from the mean on Histograms.
SPCAAllowSampDelMod	Discrete	R/W	Toggles right click menu options Delete and Modify sample options on and off.
SPCConnection	Discrete	RO	Set to 0 if the connection is lost to the server.
SPCConnectType	Message	RO	Displays whether the node is connected as a agent (Server) or Client.
SPCLowDBSpace	Discrete	RO	Used to monitor the Microsoft SQL Server database. Only works with Microsoft SQL Server databases. It will equal 1 when the database is low on space. Can be used to stop AutoCollection, and alert an operator to allocate more space or free hard disk space. It will automatically toggle between 1 and 0 depending upon the status of the SQL database.
SPCResetRunRules	Discrete	R/W	Used to reset the application of run rules on the incoming samples. This is important when collecting data for a new batch, you may not want the run rules to apply to samples from the new batch. You may want to reset the rules for the new batch.
StartCollection	Discrete	R/W	Set to 1 to start an Auto collection cycle.

All Current Sample DDE Items pertain to the last collected sample of a given dataset. They can be used to change the raw data and the limits associated with the dataset Name. To change information about the current sample, you must write to the appropriate DDE Item, then set the CurrentUpdate DDE Item to 1. This updates the sample and triggers any required calculations to be performed. The SPC program resets the CurrentUpdate DDE Item to zero after the sample has been entered. After the next sample has started a collection cycle, the current sample DDE items can no longer be updated.

Current sample DDE items are shared among all nodes. These item values represent the last collected sample of a collected product.

For distributed SPC, initially all values are set to zero. SPC connects to the database and checks for new data every five seconds. Item values are updated whenever new information is found. Modifications to the current sample values are buffered locally until the CurrentUpdate item is set to 1. Then, the values are placed in a current sample packet and sent to the remote dataset node for analysis and storage.

With the addition of displayed and collected products, all current SPC DDE items apply only to the collected product.

Item Name	DDE Type	Access	Description
CalculateCpAndCpk	Discrete	R/W	Set to 1 to calculate the Cp and Cpk values immediately after the current sample.
CurrentCauseCode	Integer	R/W	Sets the special cause code number for the current sample.
CurrentCauseString	Message (128)	RO	Displays the description of the special cause code number for the current sample.
CurrentComment	Message (50)	R/W	Used to read/write any miscellaneous comments associated with the current sample.
CurrentCp	Real	RO	Displays the capability for the current sample.

Item Name	DDE Type	Access	Description
CurrentCpk	Real	RO	Displays the centered capability for the current sample.
CurrentDate	Message (10)	R/W	Sets the Date for the current sample in the format DD/MM/YY or DD/MM/YYYY. If incorrectly entered, defaults to the current date.
CurrentFlag	Discrete	R/W	Sets a flag for the current sample.
CurrentIgnoreValue	Discrete	R/W	Sets the current sample to be ignored when the Control Chart is AutoScaled.
CurrentMx	Real	R/W	Sets the individual measurement value for the current sample. (x=1 to 25.)
CurrentNote	Message(12)	R/W	Sets the note text for the current sample on the control chart.
CurrentR	Real	RO	Displays the range for the current sample.
CurrentRBar	Real	R/W	Sets the average range at the current sample.
CurrentRLCL	Real	R/W	Sets the range Lower control limit.
CurrentRUCL	Real	R/W	Sets the range Upper control limit.
CurrentSample	Real	RO	Displays the value of the last sample point (i.e., X, C, P).
CurrentSampleBar	Real	R/W	Sets the current sample average at this sample point.
CurrentSampleNumber	Integer	RO	Displays the last sample number collected.

Item Name	DDE Type	Access	Description
CurrentTarget	Real	R/W	Sets the target value at this sample point.
CurrentTime	Message (8)	R/W	Sets the time for the current sample in the format HH:MM:SS. If incorrectly entered, defaults to the current time.
CurrentUpdate	Discrete	R/W	To change information entered about the sample in any of the current fields, set this Item to 1.
CurrentXLCL	Real	R/W	Sets current sample Lower control limit (LCL).
CurrentXLSL	Real	R/W	Sets current sample Lower Specification Limit (LSL).
CurrentXUCL	Real	R/W	Sets current sample Upper control limit (UCL).
CurrentXUSL	Real	R/W	Sets current sample Upper Specification Limit (USL).
SPC2L3Out2SD	Integer	RO	Alarm counter for Alarm "2 of the last 3 samples outside of 2 standard Deviation SS."
SPC2L3Out2SDMsg	Message	RO	Alarm message tag for "2 of the last 3 samples outside of 2 standard Deviation SS."
SPC4L5Out1SD	Integer	RO	Alarm counter for Alarm "4 of the last 5 samples outside of 1 standard Deviation SS."
SPC4L5Out1SDMsg	Message	RO	Alarm message tag for "4 of the last 5 samples outside of 1 standard Deviation SS."

Item Name	DDE Type	Access	Description
SPCConSampAltUpDn	Integer	RO	Alarm counter for Alarm "Consecutive samples Alternating Up and Down."
SPCConSampAltUpDnMsg	Message	RO	Alarm message tag for "Consecutive samples Alternating Up and Down."
SPCConSampIn1SD	Integer	RO	Alarm counter for Alarm "Consecutive samples Inside 1 standard Deviation."
SPCConSampIn1SDMsg	Message	RO	Alarm message tag for "Consecutive samples Inside 1 standard Deviation."
SPCConSampIncDec	Integer	RO	Alarm counter for Alarm "Consecutive samples Increasing or Decreasing."
SPCConSampIncDecMsg	Message	RO	Alarm message tag for "Consecutive samples Increasing or Decreasing."
SPCConSampOneSideCL	Integer	RO	Alarm counter for Alarm "Consecutive samples on one side of centerline."
SPCConSampOneSideCLMsg	Message	RO	Alarm message tag for "Consecutive samples on one side of centerline."
SPCConSampOut1SD	Integer	RO	Alarm counter for Alarm "Consecutive samples outside 1 standard Deviation."
SPCConSampOut1SDMsg	Message	RO	Alarm message tag for "Consecutive samples outside 1 standard Deviation."

Item Name	DDE Type	Access	Description
SPCNLNOOutNSD	Integer	RO	Alarm counter for Alarm "? Of the last ? samples outside ? standard deviations."
SPCNLNOOutNSDMsg	Message	RO	Alarm Message Tag for "? Of the last ? samples outside ? standard deviations."
SPCNLNOOutNSDSS	Integer	RO	Alarm counter for Alarm "? Of the last ? samples outside ? standard deviations SS."
SPCNLNOOutNSDSSMsg	Message	RO	Alarm message tag for "? Of the last ? samples outside ? standard deviations SS."
SPCOutRCtrl	Integer	RO	Alarm counter for the Range Chart Alarm "Range outside control limit."
SPCOutRCtrlMsg	Message	RO	Alarm Message for the Range Chart Alarm "Range outside control limit."
SPCOutXCtrl	Integer	RO	Alarm counter for the X Chart Alarm "Sample outside control limit."
SPCOutXCtrlMSG	Message	RO	Alarm Message for the X Chart Alarm "Sample outside control limit."
SPCOutSpec	Integer	RO	Alarm counter for Alarm "Sample outside specification Limit."
SPCOutSpecMsg	Message	RO	Alarm message tag for "Sample outside specification Limit."

Item Name	DDE Type	Access	Description
SPCRecalculateCp	Discrete	R/W	When set to 1, the engine recalculates Cp and Cpk values for the current dataset when the next sample is collected. After this sample is collected, the value of this item is reset to 0. To recalculate Cp and Cpk again, set this value to 1 again. This item ONLY affects Cp and Cpk values. It does not affect control limits or Run Rules.
SPCResetAlarmCounters	Discrete	R/W	Resets all alarm counters.
SPCResetRunRules	Discrete	R/W	Used to reset the application of run rules on the incoming samples. Applies to the current product collected. When turned on, run rules are reset and previous samples will not apply in calculations such as "4 out of 5 samples outside of 1 standard deviation" alarms. This action only occurs once and run rules resume as normal. This item must be reset and then turned again for the reset action to take place again.

The Manual Input DDE Items are used to create custom manual input windows. To use the manual input items, set the values of the appropriate items and then set the MI_Save DDE Item to 1. This will cause the information in the other MI fields to be entered as a new sample. The SPCPro program resets the MI_Save DDE Item to 0 after the sample has been entered.

For distributed SPC, manual input DDE items are private to each node. The values are buffered locally at each node until the DDE item MI_Save is set to 1. After MI_Save is set to 1, the values are placed in a manual input packet and sent to the remote dataset node for analysis and storage.

With the addition of displayed and collected products, all of the Manual SPC DDE Items apply only to the collected product.

Item Name	DDE Type	Access	Description
MI_CauseCode	Integer	WO	Sets the special cause code number for the manually input sample.
MI_CauseString	Message (127)	RO	Displays the description of the special cause code number input for the sample.
MI_Comment	Message (50)	WO	Used to read/write any miscellaneous comments entered for the sample.
MI_Date	Message (10)	WO	Sets the Date for the current sample. The Date must be entered in the format DD/MM/YY or DD/MM/YYYY. If incorrectly entered, the Date will default to the current Date.
MI_Flag	Discrete	WO	Sets a flag for the manually input sample.
MI_IgnoreValue	Discrete	WO	Sets the current sample to be ignored when the Control Chart is AutoScaled.
MI_Mx	Real	WO	Sets the value for the designated manually input measurement ($x=1$ to 25).
MI_Save	Discrete	WO	Saves the information manually entered in the other MI fields as a new sample. When the MI_Save item is set to 1, the value of all MI items are written to the respective Current DDE Items and the CurrentSampleNumber item is indexed by 1. Using MI_Save a second time will produce the same sample time as the previous usage.
MI_Time	Message (8)	WO	Sets the Time for the current sample. The Time must be entered in the format HH:MM:SS. If incorrectly entered, the Time will default to the current time.

Selection DDE Items can be used to view detailed information about any sample. The DDE Item Selection is used to enter the number of the sample to be displayed. After entering the sample number, the SPCPro updates all other Selection Items with detailed information about the sample.

Old data cannot be changed. But, special cause codes, flags, and comments can be added by setting the appropriate items, and then setting the SelectionUpdate item to 1. This causes the selection sample record to be modified with new values. SPCPro resets the SelectionUpdate DDE item to zero after the updated sample is entered.

For distributed SPC, selected sample DDE items are private to each node. They are the sample values recorded by the remote node for a specified sample number of the collected product. When the Selection DDE item is set to a sample number, the sample information is retrieved from the remote node's sample file.

Old data cannot be changed, but special cause codes, Flags, and Comments can be added by changing the appropriate DDE item and setting the SelectionUpdate item to 1. When SelectionUpdate is set to 1, the special cause code, Comment, Flag, and Ignore Value items are sent to the remote node in a packet for storage.

Note With the addition of displayed and collected products, all "Selection" SPC DDE Items apply only to the collected product.

Item Name	DDE Type	Access	Description
Selection	Integer	R/W	Setting this item to a sample number will update all of the selection items with the appropriate data for that sample.
SelectionCauseCode	Integer	R/W	Sets the special cause code number for the selected sample.
SelectionCauseString	Message (128)	RO	Displays the description of the entered special cause code.
SelectionComment	Message (50)	R/W	Used to read/write any miscellaneous comments entered for the selected sample.

Item Name	DDE Type	Access	Description
SelectionCp	Real	RO	Displays the capability for the selected sample.
SelectionCpk	Real	RO	Displays the centered capability for the selected sample.
SelectionDate	Message (10)	RO	Displays the date for the selected sample.
SelectionFlag	Discrete	R/W	Sets a Flag of the selected sample.
SelectionIgnoreValue	Discrete	R/W	Sets the selected sample to be ignored when the Control Chart is AutoScaled.
SelectionMx	Real	RO	Displays the value for the individual measurements ($x=1-25$) comprising the sample.
SelectionProduct	Message (32)	RO	Displays the Product Name for the selected sample.
SelectionRUCL	Real	RO	Displays the range UCL for the selected sample.
SelectionRLCL	Real	RO	Displays the range LCL for the selected sample.
SelectionR	Real	RO	Displays the range for the selected sample.
SelectionRBAR	Real	RO	Displays the average range at the selected sample.
SelectionSample	Real	RO	Displays the value of the selected sample point.
SelectionSampleBar	Real	RO	Displays the selected sample average at the selected sample point.
SelectionTarget	Real	RO	Displays the target value at the selected sample.
SelectionTime	Message (8)	RO	Displays the time for the selected sample.
SelectionUpdate	Discrete	R/W	Updates changes in Selection fields.

Item Name	DDE Type	Access	Description
SelectionXUSL	Real	RO	Displays sample Upper Specification Limit.
SelectionXLSL	Real	RO	Displays sample Lower Specification Limit.
SelectionXUCL	Real	RO	Displays sample Upper control limit.
SelectionXLCL	Real	RO	Displays sample Lower control limit.
SeISPC2L3Out2SDMsg	Message	RO	Alarm message tag for "2 of the last 3 samples outside of 2 standard Deviations SS."
SeISPC4L5Out1SDMsg	Message	RO	Alarm message tag for "4 of the last 5 samples outside of 1 standard Deviation SS."
SeISPCConSampAltUpDnMsg	Integer	RO	Alarm message for Alarm "Consecutive samples Alternating Up and Down."
SeISPCConSampIn1SDMsg	Message	RO	Alarm message tag for "Consecutive samples Inside 1 standard Deviation."
SeISPCConSampIncDecMsg	Message	RO	Alarm message tag for "Consecutive samples Increasing or Decreasing."
SeISPCConSampOneSideCLMsg	Message	RO	Alarm message tag for "Consecutive samples on one side of centerline."
SeISPCConSampOut1SDMsg	Message	RO	Alarm message tag for "Consecutive samples outside 1 standard Deviation."
SeISPCNLNOutNSDMsg	Message	RO	Alarm Message Tag for "? Of the last ? samples outside ? standard deviations."

Item Name	DDE Type	Access	Description
SeISPCNLNOurNSDSSMsg	Message	RO	Alarm message tag for "? Of the last ? samples outside ? standard deviations SS."
SeISPCOutRCtrlMsg	Message	RO	Alarm Message for the Range Chart Alarm "Range outside control limit."
SeISPCOutXCtrlMsg	Message	RO	Alarm Message for the X Chart Alarm "Sample outside control limit."
SeISPCOutSpecMsg	Message	RO	Alarm message tag for "Sample outside specification Limit."

Multiple Indirect datasets can be configured and linked to the same real dataset. Then, the Selection value of each Indirect dataset can be set to a different sample number. This allows you to view detail information from multiple samples within a dataset.

Adding or Deleting Datasets at Run Time

You can create QuickScripts containing the `SPCDatasetDlg()` function to add or delete a dataset.

SPCDatasetDlg() Function

The `SPCDatasetDlg()` function shows the SPCPro **Dataset Configuration** dialog box in WindowViewer. New datasets can be added or deleted from the dialog box. There are no function arguments nor a return value.

While WindowViewer is running, current dataset and product information is grayed out and these options cannot be changed.

If running in Autocollection mode, the Autocollection cycle is restarted. Restarting the Autocollection cycle results in possible data loss during the initialization phase of the datasets.

Note It is recommended that you use the NewProduct SPC DDE Item to add Products during run time, not with the `SPCDatasetDlg()` function. This prevents Autocollection mode from restarting.

Category

SPC

Syntax

```
SPCDatasetDlg ();
```

Remarks

Running a script that includes the `SPCDatasetDlg()` function opens the SPCPro **Dataset Configuration** dialog box in WindowViewer.

Example

```
SPCDatasetDlg ();
```

See Also

`SPCSelectDataset()`

Changing the Dataset Used by a Chart

Indirect datasets can be reassigned in run time to different datasets through DDE or SPC functions. This allows you to create a single SPC Chart that can show any configured dataset.

To change a chart's dataset

- 1 Create an I/O message tag. For example, **Indirect_DatasetName**.
- 2 Associate this tag to an SPCPro Access Name using SPCPro as the application name and a valid configured Indirect dataset name for the topic name.

Note Dataset names are case sensitive. If the DSN is called Indirect, and you enter indirect for the Topic in the Access Name, none of the I/O items for that Access Name will work.

- 3 In the Tagname Dictionary **Item** box, enter the SPC DDE DatasetName item name.
- 4 Create an InTouch QuickScript to change the dataset name. For example:

```
Indirect_DatasetName = "SPC1";
```

Or, create the following to allow the user to select the dataset name. For example:

```
Indirect_DatasetName = SPCSelectDataset();
```

Where: SPC1 is a valid dataset name.

After running this QuickScript, the specified dataset name (in this case, SPC1) is written to the Indirect_DatasetName I/O message tag. The SPC Chart shows the configuration settings for the specified dataset name, which include the control limits, and the last collected samples. The last collected sample is the last sample shown in the chart.

SPCSelectDataset() Function

The SPCSelectDataset() function opens the **Select a Dataset** dialog box. After the user selects a direct dataset, the function assigns the name to the DatasetName tag.

Category

SPC

Syntax

```
DatasetName=SPCSelectDataset ( )
```

Remarks

After a dataset name is selected, the function assigns the name to the *DatasetName* tag. This selection can also be used to change the *DatasetName* of an Indirect Dataset.

See Also

SPCSelectProduct(), SPCDatasetDlg()

Changing the Displayed Product in a Chart

You can dynamically change the product shown in a SPCPro chart during run time. The SPCSelectProduct() function can be used in an event-driven QuickScript to open a dialog box so that the run-time user can select a product.

To change the product within a dataset

- 1 Create an I/O message tag. For example, ProductDisplayed.
- 2 Associate this tag to an InTouch Access Name using SPCPro as the application name and a valid configured dataset name as the topic name.
- 3 In the Tagname Dictionary **Item** box, enter the SPC DDE ProductDisplayed item name for the I/O message tag.
- 4 Create an InTouch QuickScript to change the Product name. For example:

```
ProductDisplayed = "Product1";
```

In this example, Product1 is a valid product name defined in the dataset specified as the topic name in the InTouch Access Name.

Or, create the following script to allow the user to select the Product name. For example:

```
ProductDisplayed =  
SPCSelectProduct ("Dataset");
```

After running this QuickScript, the SPC chart shows the specified product.

SPCSelectProduct() Function

The SPCSelectProduct() function opens the **Select a Product** dialog box to select a product from a given dataset.

Category

SPC

Syntax

```
ProductName=SPCSelectProduct("Dataset");
```

Argument

Dataset

Contains the actual dataset name. Actual string or message tag.

Remarks

After a product name is selected, the function returns it to the ProductName tag. This name can now be used to change the collected product in the *Dataset*.

See Also

SPCSelectDataset(), SPCSetProductDisplayed(), SPCSetProductCollected()

SPCSetProductDisplayed() Function

The SPCSetProductDisplayed() function changes the product being shown in a specified dataset.

Category

SPC

Syntax

```
SPCSetProductDisplayed("Dataset", "Product");
```

Argument

Dataset

Contains the actual dataset name. Actual string or message tag.

Product

Contains the product name for which data will be shown. Actual string or message tag.

Example

```
SPCSetProductDisplayed("ADatasetName",  
    "AProductName");
```

See Also

SPCSelectProductt(), SPCSetProductCollected()

Scrolling a Chart

You can create a button and attach a **Touch Pushbutton - Action** script to scroll forward or backward through the current dataset's historical data. This is accomplished by changing the value of the SPC DDE Item `LastSampleDisplayed` in the script linked to the selected object. Another SPC DDE Item, `SamplePerControlChart`, can be used to control how much data is shown in a chart.

Note A Limit Wizard can be used to scroll forwards or backwards through the current dataset's historical data.

To scroll backwards through historical data

- 1 Create a graphic object such as a button.
- 2 Attach the following **Touch Pushbutton - Action** script to the button:

```
LastSampleDisplayed = LastSampleDisplayed -
SamplePerControlChart;
```

When the operator clicks the button during run time, the script subtracts the value of `SamplePerControlChart` from the current `LastSampleDisplayed` number. The chart scrolls backwards to show the resulting sample number as the last sample in the chart.

For example, if the current `LastSampleDisplayed` is number 860, a `SamplePerControlChart` item's value is 20, the chart scrolls backwards 20 samples. The chart shows sample number 821 as the first sample and 840 as the last chart sample.

To scroll forward through historical data

- 1 Create a graphic object such as a button.
- 2 Attach the following **Touch Pushbutton - Action** script to the object:

```
LastSampleDisplayed = LastSampleDisplayed +
SamplePerControlChart;
```

when the operator clicks the button during run time, the script adds the value of `SamplePerControlChart` to the current `LastSampleDisplayed` number. The chart scrolls forward to show the resulting sample number as the last sample in the chart. For example, if the current `LastSampleDisplayed` is number 860 and `SamplePerControlChart` item's value is 20, the chart scrolls forward 20 samples. The chart shows sample number 861 as the first sample and sample number 880 as the last sample.

Note Whenever the SPC chart is not showing the current sample number, the word "Historical" appears in the chart.

You can use three SPC functions in QuickScripts to select the data is currently shown in a chart. The following table lists these functions.

SPCDisplayData()	Scrolls the chart to a specified date and time.
SPCLocateScooter()	Moves the chart scooter to any valid sample number.
SPCMoveScooter()	Scrolls the scooter by a given number of samples.

SPCDisplayData() Function

The SPCDisplayData() function can be used in a script to scroll to a specified date and time within a chart. You can use a tag to monitor the status of the SPC data search. The tag is assigned a value of 0 if SPC found data and 1 if it could not find data for the specified time period.

Category

SPC

Syntax

```
[Status=]SPCDisplayData("Dataset", "DateString",
    "TimeString", RangeInHours);
```

Arguments

Dataset

Actual dataset name. Actual string or message tag.

DateString

Date in the format mm/dd/yyyy. Actual string or message tag.

TimeString

Time in hh:mm:ss format. Actual string or message tag.

RangeInHours

Hours of data to show in a chart. Any integer number or tag.

Example

The following example shows one hour of data for the current product of Dataset1 starting from 9:15 AM on 5/16/07.

```
StatusTag = SPCDisplayData("Dataset1",
    "05/16/2007", "09:15:00", 1);
```

SPCLocateScooter() Function

The SPCLocateScooter() function can be used in a script to move a chart Scooter by a specified number of samples. The Scooter tag defined in the dataset is updated with the X-Bar sample value. Setting SampleNumber to 0 hides/disables the Scooter.

Category

SPC

Syntax

```
SPCLocateScooter( "Dataset", SampleNumber);
```

Arguments*Dataset*

Actual dataset name. Actual string or message tag.

SampleNumber

Valid number of samples. Any number or integer tag.

See Also

SPCMoveScooter()

SPCMoveScooter() Function

The SPCMoveScooter() function can be used in a script to move a Scooter a specified number of samples within a chart. The Scooter tag defined in the dataset is updated with the X-Bar sample value.

Category

SPC

Syntax

```
SPCMoveScooter( "Dataset", IncrementValue);
```

Arguments*Dataset*

Actual dataset name. Actual string or message tag.

IncrementValue

Number of samples to scroll within the chart. Use a positive number to scroll forwards and negative number to scroll backwards. Any number or integer tag.

See Also

SPCLocateScooter()

Updating a Chart to the Current Time

You can give operators the capability to load the SPC Control Chart with current sample data during run time. Create a selectable object in your InTouch application and attach a script that updates the chart with current data.

To fill a chart with current data

- 1 Create a graphic object such as a button.
- 2 Attach the following **Touch Pushbutton - Action** script to the graphic object:

```
LastSampleDisplayed = CurrentSampleNumber;
```

In this example, CurrentSampleNumber is a SPC DDE item of the dataset.

When the operator clicks the button during run time, the script sets the LastSampleDisplayed item equal to the CurrentSampleNumber. The chart fills with previous samples according to the value of the SamplesPerControlChart item's value, including the CurrentSampleNumber. For example, if the CurrentSampleNumber value is 860 and the value of SamplesPerControlChart is 20, the chart shows sample number 841 as the first sample and 860 as the last sample.

To clear the chart (No Samples Displayed)

You can set the LastSampleDisplayed DDE Item equal to zero or any number less than the first sample number in the database. This clears the chart. Also, if the operator scrolls backwards past the first sample, the chart automatically clears.

Entering a New Sample

You can create an action QuickScript to manually enter data into an SPCPro dataset while an application is running. You use two SPCPro functions to set sample values and then save them to the dataset:

- SPCSetMeasurement() function
- SPCSaveSample() function

SPCSetMeasurement() Function

The SPCSetMeasurement() function can be used in a script to enter values to a dataset measurement number manually or on an event-driven basis.

Category

SPC

Syntax

```
SPCSetMeasurement ("Dataset", Measurement, Value) ;
```

Arguments

Dataset

Actual dataset name. String or message tag.

Measurement

Measurement number (from 1 to 300). Any number or integer tag.

Value

Value written to the specified measurement number. Any integer or real tag.

Remarks

Use the SPCSaveSample() function after all measurements are set to save the data to the database. You must re-initiate DDE conversations to update the values of the MI_Mx tags.

See Also

SPCSaveSample()

SPCSaveSample() Function

The SPCSaveSample() function saves a manual input sample. This function is used in conjunction with the SPCSetMeasurement() function.

Category

SPC

Syntax

```
SPCSaveSample ("Dataset");
```

Arguments

Dataset

Actual dataset name. String or message tag.

Remarks

Processing this function forces the sample to be entered into the specified dataset. It uses the current values of the manual input variables for the measurements in the sample. The input values are set either via the DDE tags MI_Mx (x =the measurement number form 1 to 25) or by using the SPCSetMeasurement() function (measurements 1 to 300). You must re-initiate DDE conversations to update the values of the MI_Mx tags.

See Also

SPCSetMeasurement()

Checking for Deleted Samples

You can use the `SPCSampleDeleted()` function to determine if a sample was deleted.

SPCSampleDeleted() Function

The `SPCSampleDeleted()` function returns the status of a specified sample from a dataset.

Category

SPC

Syntax

```
Result=SPCSampleDeleted("Dataset", "Product",  
    SampleNum);
```

Arguments

Dataset

Actual dataset name. String or message tag.

Product

Product name. String or message tag.

SampleNum

The sample number. Any number or integer tag.

Return Values

-1 = Invalid dataset name, product name, or sample number.

0 = Sample is not deleted.

1 = Sample is deleted.

Changing the Currently Collected Product

You can create a QuickScript that includes the `SPCSetProductCollected()` function to change the product whose data is being collected in a dataset. Alternatively, use the dataset's `ProductCollected` DDE item.

To change a dataset product

- 1 Create an I/O message tag. For example, `ProductCollected`.
- 2 Associate this tag to an InTouch Access Name using SPCPro as the application name and a valid configured dataset name for the topic name.
- 3 In the Tagname Dictionary **Item** box, enter the SPC DDE `ProductCollected` Item Name for the `ProductCollected` tag.

- 4 Create an InTouch QuickScript to change the Product name. For example:

```
ProductCollected = "Product1";
```

In this example, `Product1` is a valid Product name defined in the dataset specified as the topic name in the InTouch Access Name.

Or, create the following script to allow the user to select the Product name. For example:

```
ProductCollected =  
SPCSelectProduct("Dataset");
```

After running this QuickScript, the specified Product name is written to the `ProductCollected` I/O message tag.

SPCSetProductCollected() Function

The SPCSetProductCollected() function changes the product being collected in a specified dataset.

Category

SPC

Syntax

```
SPCSetProductCollected("Dataset", "Product");
```

Arguments

Dataset

Actual dataset name. String or message tag.

Product

Product name under which data should be collected. String or message tag.

Remarks

The SPCSetProductCollected() function does not change the product shown in a chart. It is possible to collect data for one product and show data for another by using this function to collect and the SPCSetProductDisplayed() function to show collected data.

Examples

```
SPCSetProductCollected("Data5838", "Widgets");
```

See Also

SPCSelectProduct(), SPCSetProductDisplayed()

Creating or Deleting Products During Run Time

The SPCPro program provides a DDE item and functions that allow you to create or delete a product within an existing dataset. Creating new product names can also be used to create separate product files for new lot numbers of an existing product.

- NewProduct DDE Item
- SPCCreateNewProduct() Function
- SPCDeleteProduct() Function

NewProduct DDE Item

You can create a new product by setting the NewProduct SPC DDE item.

To create a new product in run time

- 1 Create an I/O message tag. For example, NewProduct.
- 2 Associate this tag to an InTouch Access Name using SPCPro as the application name and a valid dataset name for the topic name.
- 3 In the **Item** box of the Tagname Dictionary, enter the NewProduct SPC DDE Item Name for the NewProduct tag.

- 4 Create an InTouch QuickScript to change the Product name. For example:

```
NewProduct = "Product2";
```

In this example, Product2 is not an existing product name defined in the dataset specified as the topic name in the InTouch Access Name.

- 5 Set the new products, Control Chart, Pareto Chart, and Histogram Title by using these DDE items:

```
NewProductCtrlTitle  
NewProductParetoTitle  
NewProductHistTitle
```

If these are set before you call NewProduct, the newly created product uses the new title names. For example:

```
NewProductCtrlTitle = "Product2";  
NewProductParetoTitle= "Product2";  
NewProductHistTitle= "Product2";  
NewProduct= "Product2";
```

When this script runs, the specified product name is written to the NewProduct I/O message tag. SPCPro automatically sets the value of the ProductCollected SPC DDE Item to the value of NewProduct.

SPCCreateNewProduct() Function

The SPCCreateNewProduct() function creates a new product.

Category

SPC

Syntax

```
SPCCreateNewProduct ("Dataset", "Product",  
    MeasPerSample);
```

Parameters

Dataset

The dataset name. An actual string or a message tag.

Product

The name of the product to create. An actual string or message tag.

MeasPerSample

The measurements per sample for the product. Any number or an integer tag.

See Also

SPCDeleteProduct()

SPCDeleteProduct() Function

The SPCDeleteProduct() function deletes all product information from the SPC database, including the product's measurements and samples.

Category

SPC

Syntax

```
SPCDeleteProduct ("Dataset", "Product");
```

Parameters

Dataset

The dataset name. An actual string or a message tag.

Product

The name of the product to delete. An actual string or message tag.

Remarks

If the product is currently collected or shown, it cannot be deleted. A message is logged to the Log Viewer.

See Also

SPCCreateNewProduct()

Entering Calculation Parameters for Various Chart Types

You can create QuickScripts to enter calculation values for SPCPro control, range, or specification limits. Use these functions to enter calculation parameters:

- SPCSetControlLimits() Function
- SPCSetRangeLimits() Function
- SPCSetSpecLimits() Function
- SPCSetProductControlLimits() Function
- SPCSetProductRangeLimits() Function
- SPCSetProductSpecLimits() Function

SPCSetControlLimits() Function

The SPCSetControlLimits() function can be used in a script to enter control limit values for a Control Chart manually or by an event.

Category

SPC

Syntax

```
SPCSetControlLimits("Dataset", XUCL, XLCL);
```

Arguments

Dataset

Actual dataset name. String or message tag.

XUCL

Value used for the UCL of the chart. Number or real tag.

XLCL

Value used for the LCL of the chart. Number or real tag.

Remarks

These measurements are saved to the sample by processing SPCSaveSample().

See Also

SPCSaveSample(), SPCSetRangeLimits(), SPCSetSpecLimits(), SPCSetMeasurement()

SPCSetRangeLimits() Function

The SPCSetRangeLimits() function can be used in a script to set the Control Limits of a Range Chart. The SPCSetRangeLimits() function can be used for manual or event driven input of the Control Limits for a Range Chart.

Category

SPC

Syntax

```
SPCSetRangeLimits ("Dataset", RUCL, RLCL);
```

Arguments*Dataset*

Actual dataset name. String or message tag.

RUCL

Value used for the UCL for a Range Chart. Number or real tag.

RLCL

Value used for the LCL for a Range Chart. Number or real tag.

See Also

SPCSetControlLimits(), SPCSetSpecLimits()

SPCSetSpecLimits() Function

The SPCSetSpecLimits() function can be used in a script for manual or event driven input of the specification limit values of a Control Chart.

Category

SPC

Syntax

```
SPCSetSpecLimits ("Dataset", XUSL, XLSL);
```

Arguments*Dataset*

Actual dataset name. String or message tag.

XUSL

Value used for the chart's USL. Number or real tag.

XLSL

Value used for the chart's LSL. Number or real tag.

See Also

SPCSetControlLimits(), SPCSetRangeLimits()

SPCSetProductControlLimits() Function

The SPCSetProductControlLimits() function can be used in a script to enter control limit values for a Control Chart for an empty product.

Category

SPC

Syntax

```
SPCSetProductControlLimits("Dataset", "Product",  
    XUCL, XLCL);
```

Arguments

Dataset

Actual dataset name. String or message tag.

Product

Name of the product whose control limits are to be set.
String or message tag.

XUCL

Value used for the UCL of the chart. Number or real tag.

XLCL

Value used for the LCL of the chart. Number or real tag.

Remarks

These measurements are saved to the sample by the SPCSaveSample() function.

See Also

SPCSaveSample(), SPCSetProductRangeLimits(),
SPCSetProductSpecLimits(),

SPCSetProductRangeLimits() Function

The SPCSetProductRangeLimits() function can be used in a script to set the Control Limits of a Range Chart for an empty product.

Category

SPC

Syntax

```
SPCSetProductRangeLimits ("Dataset", "Product",  
    UCL, LCL);
```

Arguments

Dataset

Actual dataset name. String or message tag.

Product

Name of the product whose range limits are to be set.
String or message tag.

RUCL

Value used for the UCL for a Range Chart. Number or real tag.

RLCL

Value used for the LCL for a Range Chart. Number or real tag.

See Also

SPCSetProductControlLimits(), SPCSetProductSpecLimits()

SPCSetProductSpecLimits() Function

The SPCSetProductSpecLimits() function can be used in a script to set the specification limit values of a Control Chart for an empty product.

Category

SPC

Syntax

```
SPCSetProductSpecLimits("Dataset", "Product",  
    XUSL, XLSL);
```

Arguments

Dataset

Actual dataset name. String or message tag.

Product

Name of the product whose specification limit values are to be set. String or message tag.

XUSL

Value used for the chart's USL. Number or real tag.

XLSL

Value used for the chart's LSL. Number or real tag.

See Also

SPCSetProductControlLimits(),
SPCSetProductRangeLimits()

Entering Data into Attribute Charts

Use the following DDE items to enter data into attribute charts:

Chart Type	DDE Items Used	Comment
C-Chart	MI_M1	Enter the reject count.
P-Chart	MI_M1 and MI_M2	Enter the number of rejects and sample size. The scale range is 0.00 to 1.00. Example 0.50 = 50% rejects.
NP-Chart	MI_M1	Enter the number of defects (Set sample size in DS Configuration Dialog).
U-Chart	MI_M1 and MI_M2	Enter the Defects in MI_M1 and the sample size in MI_M2.

Controlling Configuration Options

The SPCDatasetDlg() function shows the SPCPro **Dataset Configuration** dialog box in WindowViewer. New datasets can be added or deleted from the SPCPro **Dataset Configuration** dialog box during run time. For more information, see SPCDatasetDlg() Function on page 65.

Migrating SPC Databases from a Previous SPCPro Version

You can use the SPCPro Server utility to perform all SPCPro functions. For example, configure a database, configure users, and create datasets and Indirect datasets. By using the SPCPro Server utility, you do not need to configure SPC from within WindowMaker.

You must create a new database to store your converted dataset files. The database you create must be empty. You can only import existing datasets to an empty database. Tags must be defined in the Tagname Dictionary before storing dataset information to a database.

To import SPC datasets

- 1 If needed, stop WindowMaker and WindowViewer.
- 2 Start the SPCPro utility from the command prompt or Windows Explorer. The SPCPro **Server** dialog box appears.
- 3 On the **Database** menu, click **Import**. The **Open** dialog box appears.
- 4 Double-click the application directory for the SPCPro application whose datasets you want to convert or select it, and then click **Open**. The **Import Datasets** dialog box appears.
- 5 In the **Import Options** area, select the **Select All Datasets** option to import all datasets from the selected application or, **Select All Products** to import all products from the selected application or, pick and choose the datasets and products from the listing on the left.
- 6 In the **Import Options** area, Select what samples you want to import in the second **Import Options** group as follows:

Import No Samples	Import only the dataset configurations.
Import All Samples	Import all samples in the dataset.
Import Range of Samples	Import only the samples for the specified From Date and To Date .

- 7 Click **Import**. The datasets are imported and converted to the new SPCPro format and can be viewed in SPCPro.

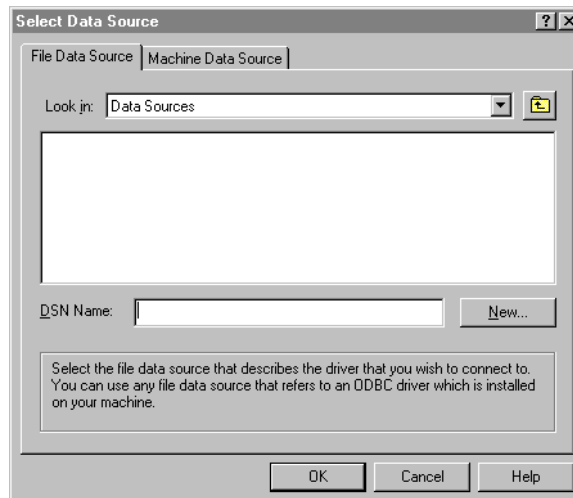
Any dataset that you are using with SPCPro version 6.0 or earlier, must be converted to the new format before running your InTouch application.

You use the SPCPro utility to upgrade SPCPro databases to the current version supported by SPCPro. The SPCPro utility is located in the same folder where InTouch is installed.

When you upgrade a database, the SPCPro utility creates a backup of the database. The backup is saved as a file in the same folder where the database is located. You can delete the database backup file after you verify the upgraded database is working correctly.

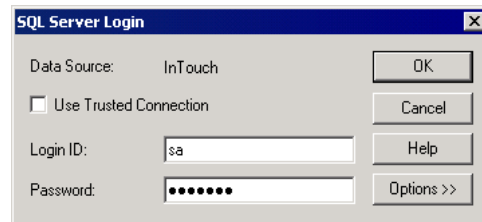
To upgrade to SPCPro database to a newer version

- 1 Verify that SPCPro is not running.
- 2 Run the `sputil.exe` command from Windows Explorer or the Windows command prompt. The **sputil** dialog box appears.
- 3 On the **File** menu, select **Convert Schema**. The **Select Data Source** dialog box appears.



- 4 Select your data source name by one of the following methods:
 - In the **DSN Name** box, type the data source name that points to the database you want to convert and click **OK**.
 - Click the **Machine Data Source** tab, select the data source name from the list, and click **OK**.

The **SQL Server Login** dialog box appears.



- 5 Log on to SQL Server.
 - a In the **Login ID** box, type the login ID of a SQL Server administrator.
 - b In the **Password** box, type the administrator's password.
 - c Click **OK**.

SPCPro begins upgrading the database. A message appears, which indicates the database upgrade is finished.

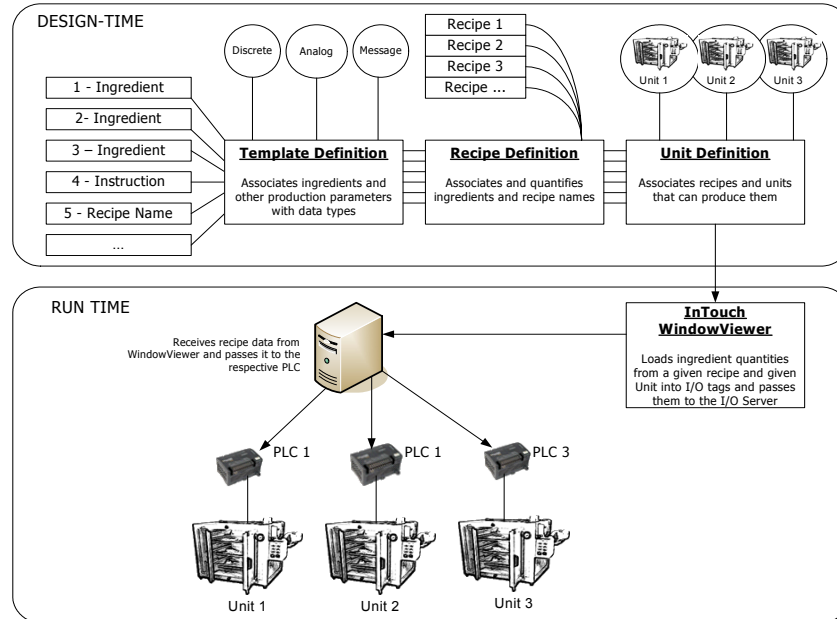
- 6 Click **OK**.

Chapter 2

Using Recipe Manager

Manufacturing industries build products according to repeatable procedures that use standardized quantities of raw materials. In essence, products are manufactured according to *recipes*. A recipe describes the raw materials, their quantities, and how they are combined to produce a finished product. In the most intuitive case, a bakery may follow a basic recipe that lists all ingredients and procedural steps to make cookies.

Recipe Manager is a supplementary component for the InTouch HMI that you can use to simplify the process of creating manufacturing recipes. The following figure summarizes how Recipe Manager obtains information from recipe templates to manage a process that creates a product.



Overview of Recipe Manager

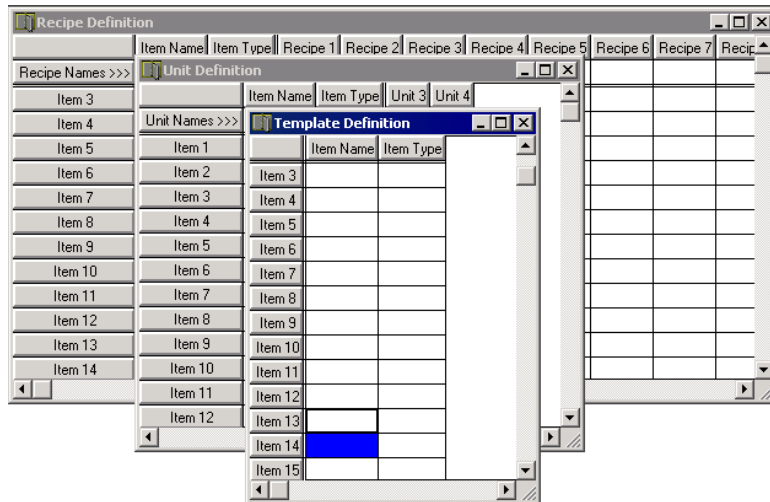
Recipe Manager can be installed with InTouch as an optional component. Recipe Manager consists of the Recipe Manager utility and a set of InTouch recipe script functions.

You can access the Recipe Manager utility from WindowMaker or from the Windows Start Menu. The Recipe Manager utility includes an interface for you to create and edit recipe templates. Recipe Manager saves your templates in a recipe file.

Typically, tags associated with a manufacturing process use QuickScripts to access data within recipe template files. Recipe Manager includes a set of QuickScript functions to select, load, modify, create, and delete the manufacturing recipes contained in template files.

Recipe Manager Utility

The Recipe Manager utility provides a spreadsheet-like user interface to create and maintain a recipe template file. A file consists of three templates. You create and edit these templates by adding or modifying data within the cells of each template's spreadsheet.



You save these templates to a Comma Separated Value (CSV) file. You can create and edit your recipe template definitions with any program that supports the .csv file format like Notepad or Excel. But, Recipe Manager provides preformatted spreadsheets and a set of editing tools to create and maintain templates reliably and easily.

Recipe Template Files

A Recipe Manager template file contains the following information:

- Names of ingredients and their data types used in a recipe.
- Unit names that associate InTouch tags with recipe ingredient values.
- Recipe names containing the quantities or values for each ingredient used in a recipe instance.

Template Definition

The Template Definition template defines all recipe ingredients. A data type is associated with each recipe ingredient. An ingredient data type can be analog, discrete, or message. Ingredient names are not required to be InTouch tags.

Unit Definition

The Unit Definition template associates InTouch tags with recipe ingredients. Many different loading definitions can be created. These definitions are called units. You can use the RecipeLoad() function to load specific instances of a recipe to associated InTouch tags. A Unit Definition can consist of all ingredients defined in the file or just a subset of these ingredients.

Note Unit tags can be memory types that can be viewed and edited in an InTouch window or I/O tags that can be loaded directly to PLCs.

Recipe Definition

The Recipe Definition template specifies the name of each recipe and ingredient quantities used by the recipe. Recipe instances can be modified, created, or deleted in run time through the recipe functions.

Editing Recipe Data in Recipe Manager

You create manufacturing recipes by completing a set of sequential tasks. The following list shows the Recipe Manager tasks to create recipes and the order in which they should be completed:

- Configuring the Recipe Manager editing grid.
- Editing data within a template.
- Assigning ingredient names and unit types to the Template Definition template.
- Mapping InTouch tags to ingredients in the Unit Definition template.
- Assigning values to recipe ingredients in the Recipe Definition template.

Configuring the Recipe Manager Editing Grid

Before you create manufacturing recipes, you should configure Recipe Manager. There are two tasks to configure Recipe Manager editing functions:

- Set the maximum limit for template items.
- Set the ENTER key scroll function.

Before you create recipes, you need to configure the maximum number of items that can be entered in your recipe templates. You must assign a set of maximum limits for items, units, and recipe names.

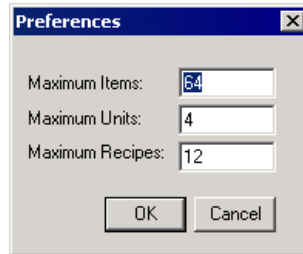
Templates can contain up to a maximum of 9999 items, units, and recipe names. However, large maximum limits can potentially affect system performance. Also, you may see an error message if the maximum limits you set will require more memory than the computer's available memory.

To configure recipe template maximum limits

- 1 Start Recipe Manager by one of the following methods:
 - Start WindowMaker. On the **Tools** view, expand **Applications**, and then select Recipe Manager.
 - Click **Start** from your Windows desktop. Click in order **Programs, Wonderware, InTouch**, and finally Recipe Manager.

The Recipe Manager dialog box appears.

- 2 On the **Options** menu, click **Preferences**. The **Preferences** dialog box appears.



- 3 In the **Maximum Items** box, enter the maximum number of item names allowed in your **Template Definition** template.
- 4 In the **Maximum Units** box, enter the maximum number of units allowed in your **Unit Definition** template.
- 5 In the **Maximum Recipes** box, enter the maximum number of recipe names allowed in your **Recipe Definition** template.

Caution The values you set in the **Preferences** dialog box are applied to all recipe template files that you create. When you modify these values, all existing recipe template files are also modified.

- 6 Click **OK**.

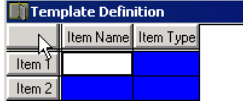
Recipe Manager includes an option that simplifies entering data in recipe templates. When you select the **Auto Down on [Enter]** option, you can press ENTER to move the cursor down to the next cell in the column.

To set ENTER key template scrolling

- 1 Open Recipe Manager.
By default, Recipe Manager does not scroll automatically to the next cell in the template spreadsheet.
- 2 On the **Options** menu, click **Auto Down on [Enter]** to set cell scrolling.
- 3 Click **Auto Down on [Enter]** again if you want to turn off cell scrolling.

Working with the Editing Grid

Recipe Manager includes a set of editing commands to add, modify, or delete data from the templates. Generally, you select the data that you want to edit in the template, and then take an editing action. The following table describes common features of recipe templates to enter and select template data.

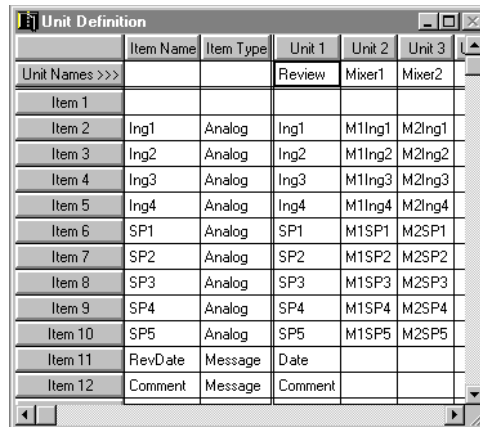
Feature	Description
Input Box	Text input box used to enter data for the selected template cell. When a template cell is selected, its contents are shown in the text input box near the top of the Recipe Manager dialog box.
Select All Cells	Click the top left cell of the template to select all cells. 
Select All Rows	Click on a template's row name to select all cells within the row.
Select All Columns	Click on a template's column heading to select all cells within the column.
Auto-Size All Columns	Double-click on a template to auto-size all columns in the template to the width of the longest entry.
Auto-Size a Column	Double-click on the heading to auto-size the column to the width of its longest entry. <p>Note The Item Type column in the Template Definition template cannot be auto-sized.</p>

When you edit a template, you can do the following:

- Clear data from a range of cells.
- Copy data from a range of cells to an adjacent range of selected cells.
- Insert a row within the **Template Definition** template.
- Insert a column within a template.
- Delete a row from the **Template Definition** template.
- Delete a column from a template.

To clear data from a range of cells

- 1 Select a range of cells from the template.

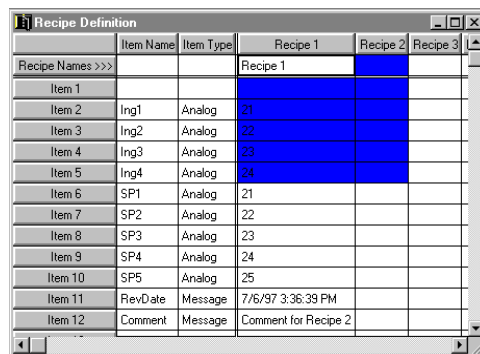


Unit Names >>>	Item Name	Item Type	Unit 1	Unit 2	Unit 3
			Review	Mixer1	Mixer2
Item 1					
Item 2	Ing1	Analog	Ing1	M1Ing1	M2Ing1
Item 3	Ing2	Analog	Ing2	M1Ing2	M2Ing2
Item 4	Ing3	Analog	Ing3	M1Ing3	M2Ing3
Item 5	Ing4	Analog	Ing4	M1Ing4	M2Ing4
Item 6	SP1	Analog	SP1	M1SP1	M2SP1
Item 7	SP2	Analog	SP2	M1SP2	M2SP2
Item 8	SP3	Analog	SP3	M1SP3	M2SP3
Item 9	SP4	Analog	SP4	M1SP4	M2SP4
Item 10	SP5	Analog	SP5	M1SP5	M2SP5
Item 11	RevDate	Message	Date		
Item 12	Comment	Message	Comment		

- 2 On the **Edit** menu, click **Clear**. A message appears requesting confirmation that the selected range of cells should be cleared.
- 3 Click **Yes**. The template clears data from the selected range of cells.

To copy a range of cells to an adjacent selected range

- 1 Select the cell or the range of cells to be copied.
- 2 Select the adjacent range of cells that you want to copy the data.



Recipe Names >>>	Item Name	Item Type	Recipe 1	Recipe 2	Recipe 3
			Recipe 1		
Item 1					
Item 2	Ing1	Analog	21		
Item 3	Ing2	Analog	22		
Item 4	Ing3	Analog	23		
Item 5	Ing4	Analog	24		
Item 6	SP1	Analog	21		
Item 7	SP2	Analog	22		
Item 8	SP3	Analog	23		
Item 9	SP4	Analog	24		
Item 10	SP5	Analog	25		
Item 11	RevDate	Message	7/6/97 3:36:39 PM		
Item 12	Comment	Message	Comment for Recipe 2		

The selected ranges must be the same size and can be above, below, to the right, or to the left of the original range of selected cells.

- 3 On the **Edit** menu, select the appropriate fill command. The data is copied to the selected range of cells.

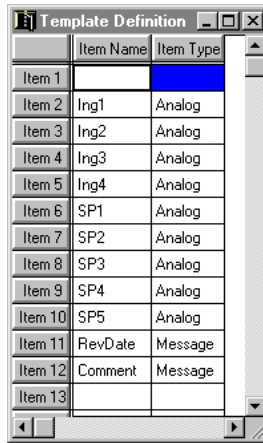
Note If the new column that the data is copied to is not big enough to accommodate the largest entry, double-click on the column heading to change the width to the longest entry.

To insert a row in the Template Definition template

- 1 Select the **Template Definition** template.
- 2 Click the **Item #** to select the row in the **Template Definition** template to insert a new row above it.

You cannot directly insert rows in either the **Recipe Definition** or **Unit Definition** templates. Instead, all modifications to the **Template Definition** are automatically inherited by the **Recipe Definition** and **Unit Definition** templates.

- 3 On the **Edit** menu, click **Insert**. A new row is inserted immediately above the row you selected and all subsequent rows are automatically renumbered.



	Item Name	Item Type
Item 1		
Item 2	Ing1	Analog
Item 3	Ing2	Analog
Item 4	Ing3	Analog
Item 5	Ing4	Analog
Item 6	SP1	Analog
Item 7	SP2	Analog
Item 8	SP3	Analog
Item 9	SP4	Analog
Item 10	SP5	Analog
Item 11	RevDate	Message
Item 12	Comment	Message
Item 13		

Note If the maximum values configured for the Recipe Manager **Preferences** have been reached, the **Insert** command is inactive. You must increase the numbers specified to add Items/Units/Recipes to your recipe templates.

When you modify the **Preferences**, the changes are applied to all existing recipe template files.

To insert a column

- 1 Click **Unit #** or **Recipe #** to select the column that will be to the right of the inserted column.

You can insert columns in the **Recipe Definition** or **Unit Definition** templates.

- 2 On the **Edit** menu, click **Insert**. A new column is inserted to the left of the selected column.

	Item Name	Item Type	Unit 1	Unit 2	Unit 3
Unit Names >>>			Review		Mixer2
Item 1					
Item 2	Ing1	Analog	Ing1	M2Ing1	
Item 3	Ing2	Analog	Ing2	M2Ing2	
Item 4	Ing3	Analog	Ing3	M2Ing3	
Item 5	Ing4	Analog	Ing4	M2Ing4	
Item 6	SP1	Analog	SP1	M2SP1	
Item 7	SP2	Analog	SP2	M2SP2	
Item 8	SP3	Analog	SP3	M2SP3	
Item 9	SP4	Analog	SP4	M2SP4	
Item 10	SP5	Analog	SP5	M2SP5	
Item 11	RevDate	Message	Date		
Item 12	Comment	Message	Comment		

In this example, **Mixer2** data is moved to the **Unit 3** column and a new column inserted as **Unit 2**.

To delete a column

- 1 Click the **Unit #** or **Recipe #** column heading to select the column that you want to delete.

You can delete columns from the **Recipe Definition** or **Unit Definition** templates.

- 2 On the **Edit** menu, click **Delete**. A confirmation message dialog box appears asking you to confirm the deletion.
- 3 Click **Yes**. The column is deleted from the template and the remaining columns are renumbered.

To delete a row

- 1 Select the **Template Definition** template.

You can delete rows from the **Template Definition** template, but not the **Recipe Definition** or **Unit Definition** templates.

- 2 Click the **Item #** row header to select the row that you want to delete.
- 3 On the **Edit** menu, click **Delete**. A confirmation message dialog box appears asking you to confirm the deletion.
- 4 Click **Yes**. The row is deleted from the template.

Defining Ingredient Names and Data Types

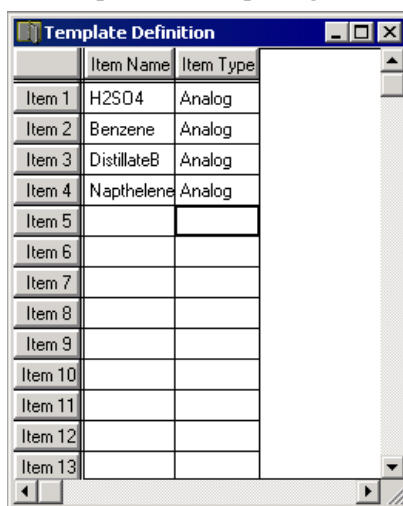
The Template Definition template lists recipe ingredients and the unit type associated with each ingredient. You must complete the Template Definition template first before adding data to the other recipe templates.

To define a Template Definition template

- 1 Start Recipe Manager.
- 2 On the **File** menu, click **New**. The three Recipe Manager templates appear.
- 3 Click the **Template Definition** title bar to select the template window.
- 4 In the **Item Name** column cells, type the names you selected for recipe ingredients.

You can only type one ingredient per cell.

- 5 In the **Item Type** column cells, type a valid item type for the respective recipe ingredient.



	Item Name	Item Type
Item 1	H2SO4	Analog
Item 2	Benzene	Analog
Item 3	DistillateB	Analog
Item 4	Naphthelene	Analog
Item 5		
Item 6		
Item 7		
Item 8		
Item 9		
Item 10		
Item 11		
Item 12		
Item 13		

Valid item types are; analog, discrete, or message. Type A for analog, D for discrete, or M for message. Recipe Manager automatically completes the remainder of the item type when you press ENTER.

Mapping InTouch Tags to Ingredients

The **Unit Definition** template associates InTouch tags with recipe ingredients for given units. As shown in the following figure, the first two columns of the **Unit Definition** template list the Item Names and Item Types from the **Template Definition** template.

	Item Name	Item Type	Unit 1	Unit 2	Unit 3	Unit 4
Unit Names >>>						
Item 1	H2SO4	Analog				
Item 2	Benzene	Analog				
Item 3	DistillateB	Analog				
Item 4	Naphthelene	Analog				
Item 5						
Item 6						
Item 7						
Item 8						
Item 9						
Item 10						
Item 11						
Item 12						

The tags defined for a unit can be memory tags or remote tags that obtain PLC data from an I/O Server.

When you use the `RecipeLoad()` function in an InTouch QuickScript, you must specify a Unit Name. The values contained in that Recipe Name definition are then loaded into the tags specified in the Unit Name when the QuickScript runs.

To define a Unit Definition template

- 1 Click the **Unit Definition** template's title bar to select the template window.

	Item Name	Item Type	Unit 1	Unit 2	Unit 3
Unit Names >>>			Review	Mixer 1	Mixer 2
Item 1	Ing1	Analog	Ing1	M1Ing1	M2Ing1
Item 2	Ing2	Analog	Ing2	M1Ing2	M2Ing2
Item 3	Ing3	Analog	Ing3	M1Ing3	M2Ing3
Item 4	Ing4	Analog	Ing4	M1Ing4	M2Ing4
Item 5	SP1	Analog	SP1	M1SP1	M2SP1
Item 6	SP2	Analog	SP2	M1SP2	M2SP2
Item 7	SP3	Analog	SP3	M1SP3	M2SP3
Item 8	SP4	Analog	SP4	M1SP4	M2SP4
Item 9	SP5	Analog	SP5	M1SP5	M2SP5
Item 10	RevDate	Message	Date		
Item 11	Comment	Message	Comment		

- 2 In the **Unit Names** row, type the name of each unit that you want to define.

- 3 In the **Unit #** column cells, use one of the following methods to enter the name of the InTouch tag for each respective recipe ingredient:
 - Type the tag name.
 - If WindowMaker is running, double-click the cell to display the **Select Tag** dialog box. Then, double-click the desired tag to insert it into the cell or select it, and then click **OK**.
- 4 Repeat this procedure for each Unit/Recipe combination.

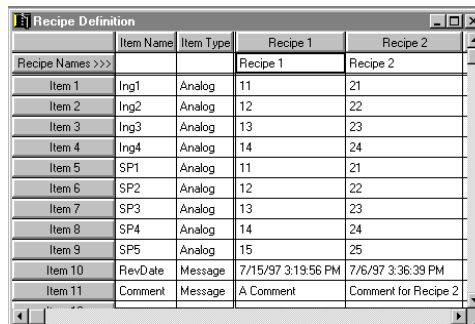
Defining Values for Ingredients in Different Recipes

The Recipe Definition template specifies the name of each recipe and ingredient quantities used by the recipe. The Recipe Definition template displays the Item Name and Item Type information from the previously defined Template Definition template.

Ingredient values are loaded into the InTouch tags when the **RecipeLoad()** function is executed in an InTouch QuickScript.

To define a Recipe Definition template

- 1 Click the **Recipe Definition** template's title bar to select the template window.
- 2 In the **Recipe Names** row, type the name of each recipe that you want to define.

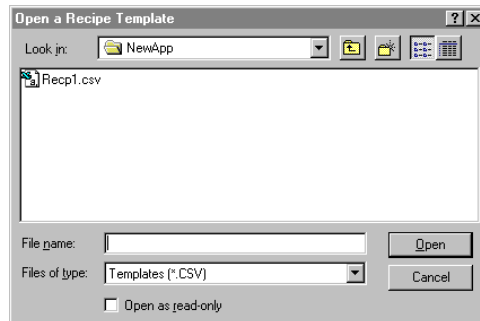


Recipe Definition				
	Item Name	Item Type	Recipe 1	Recipe 2
Recipe Names >>>			Recipe 1	Recipe 2
Item 1	Ing1	Analog	11	21
Item 2	Ing2	Analog	12	22
Item 3	Ing3	Analog	13	23
Item 4	Ing4	Analog	14	24
Item 5	SP1	Analog	11	21
Item 6	SP2	Analog	12	22
Item 7	SP3	Analog	13	23
Item 8	SP4	Analog	14	24
Item 9	SP5	Analog	15	25
Item 10	RevDate	Message	7/15/97 3:19:56 PM	7/6/97 3:36:39 PM
Item 11	Comment	Message	A Comment	Comment for Recipe 2

- 3 In the **Recipe #** column cells, type the values for each respective recipe ingredient in the **Item Name** column.
- 4 On the **File** menu, click **Save** to save your recipe template file.

To open an existing recipe template file

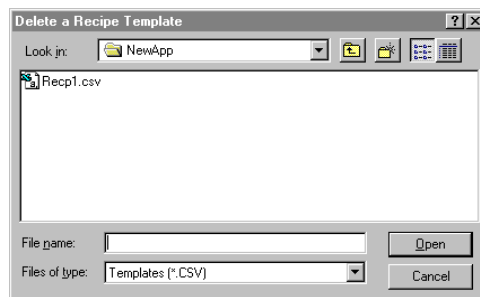
- 1 Open Recipe Manager.
- 2 On the **File** menu, click **Open**. The **Open a Recipe Template** dialog box appears.



- 3 Locate and select the Recipe file, then click **Open**. The three recipe templates in the file appear within Recipe Manager for editing.

To delete a recipe template file

- 1 On the **File** menu, click **Delete**. The **Delete a Recipe Template** dialog box appears.



- 2 Locate and select the recipe file, then click **Open** or double-click the file name. A message box appears asking you to confirm the deletion.

Note Open recipe template files cannot be deleted.

- 3 Click **Yes** to delete the file.

Editing Recipe Data in Other Applications

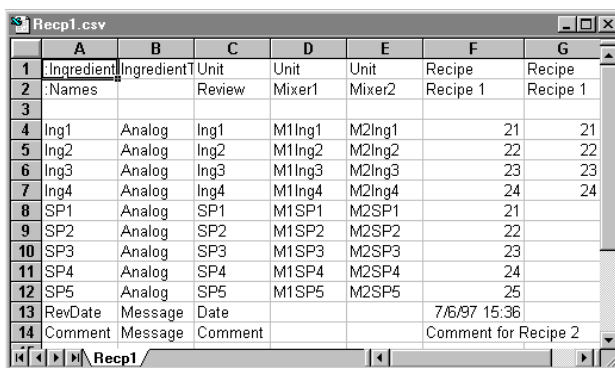
You can create and edit your recipe template definitions with any program that supports comma delimited data. You can use Microsoft Excel or Notepad to create and edit the Recipe Manager template file.

Using Excel with a Recipe Template File

You can use Excel to create or edit a recipe template if you do not want to use the Recipe Manager utility. You must save the Recipe Manager template created or edited with Excel to a file with a .csv file name extension.

To open an existing recipe template file in Microsoft Excel

- 1 Start Excel.
- 2 On the **File** menu, click **Open**. The **Open** dialog box appear.
- 3 Locate and select the .csv file then, click **Open** or, double-click the file name. Excel shows the contents of the file.



	A	B	C	D	E	F	G
1	Ingredient1	Ingredient1	Unit	Unit	Unit	Recipe	Recipe
2	Names		Review	Mixer1	Mixer2	Recipe 1	Recipe 1
3							
4	Ing1	Analog	Ing1	M1Ing1	M2Ing1	21	21
5	Ing2	Analog	Ing2	M1Ing2	M2Ing2	22	22
6	Ing3	Analog	Ing3	M1Ing3	M2Ing3	23	23
7	Ing4	Analog	Ing4	M1Ing4	M2Ing4	24	24
8	SP1	Analog	SP1	M1SP1	M2SP1	21	
9	SP2	Analog	SP2	M1SP2	M2SP2	22	
10	SP3	Analog	SP3	M1SP3	M2SP3	23	
11	SP4	Analog	SP4	M1SP4	M2SP4	24	
12	SP5	Analog	SP5	M1SP5	M2SP5	25	
13	RevDate	Message	Date			7/6/97 15:36	
14	Comment	Message	Comment			Comment for Recipe 2	

- 4 Edit the contents of the recipe file and save your changes.

To create a new recipe template file in Excel

- 1 Start Excel.
- 2 Create a new workbook.

- Enter recipe data in the spreadsheet, as shown in the following figure.

	A	B	C	D	E	F	G
1	IngredientName	IngredientType	Unit	Unit	Unit	Unit	Recipe
2	Names		Review	Mixer 1	Mixer 2	Mixer 3	Recipe 1
3	Ing1	Analog	Ing1	M1Ing1	M2Ing1	M3Ing1	11
4							
5							
6							
7							
8							

The entries must be made in the order shown in the figure. Unit Names must be defined in columns to the left of columns containing Recipe Names.

- Save the spreadsheet with a .csv file name extension.

Using Notepad with a Recipe Template File

You can use Notepad to create or edit a recipe template if you do not want to use the Recipe Manager utility. You must save the Recipe Manager template created or edited with Notepad to a file with a .csv file name extension.

To open an existing recipe template file in Notepad

- Start Notepad.
- On the **File** menu, click **Open**. The **Open** dialog box appear.
- Locate and select the recipe file, then click **Open** or double-click the file name.
- Edit the contents of the recipe file and save your changes.

To create a new recipe template file in Notepad

- Start up Notepad.
- On the **File** menu click **New**.
- Enter following data in this format:


```
:IngredientName, IngredientType [, Unit] ... [, Recipe] ...
:Names, , [, UnitName] ... [, RecipeName] ...
IngredientName, {Analog, Discrete, Message}, [, tag] ... [, value]
```

Note All Unit Names must be defined in the file before any Recipe Names are defined.

- Save the file with a .csv file name extension.

Nesting Recipes to Create Complex Structures

Multiple recipe template files can be linked to each other with InTouch QuickScripts to create complex applications. You link recipe template files by defining an ingredient name that is associated with a message tag in the Unit Name template. Then, you load the message tag with the name of a recipe.

Linking recipe template files allows you to create master recipe template files that define machine configuration parameters used by various recipes in different recipe files. Keeping this type of information in one central file greatly reduces the time it takes to maintain and update data whenever it changes.

In the following figure, the Item Name **Setup** tag is defined as a message type and the units contain the **Setup** message tag for this item. Each recipe contains a second recipe name defined in a different recipe file that is loaded into the **Setup** tag when the recipe is selected.

RECFILEA.CSV									
1	2	3	4	5	6	7	8	9	
Item Name	Item Type	Unit	Unit	Unit	Unit	Recipe	Recipe	Recipe	
2	:Names	Review	Mixer 1	Mixer 2	Mixer 3	Recipe 1	Recipe 2	Recipe 3	
3	Ing1	Analog	Ing1	M1Ing1	M2Ing1	M3Ing1	11	21	31
4	Ing2	Analog	Ing2	M1Ing2	M2Ing2	M3Ing2	12	22	99
5	Ing3	Analog	Ing3	M1Ing3	M2Ing3	M3Ing3	13	23	66
6	Ing4	Analog	Ing4	M1Ing4	M2Ing4	M3Ing4	14	24	34
7	SP1	Analog	SP1	M1SP1	M2SP1	M3SP1	11	21	31
8	SP2	Analog	SP2	M1SP2	M2SP2	M3SP2	12	22	32
9	SP3	Analog	SP3	M1SP3	M2SP3	M3SP3	13	23	33
10	SP4	Analog	SP4	M1SP4	M2SP4	M3SP4	14	24	34
11	SP5	Analog	SP5	M1SP5	M2SP5	M3SP5	15	25	35
12	Setup	Message	Setup	LinkFile	LinkFile	LinkFile	Setup2A	Setup3A	Setup1A
13									

To do so, the following script would be entered:

```
RecipeName="Recipe2";
RecipeLoad("c:\recipe\recfilea.csv", "Review",
  RecipeName);
```

When this script runs, the value of the **Setup** tag becomes Setup3A and is loaded into the **Review** unit. The value of the **Setup** tag is then used as the Recipe Name in the next recipe loading that loads the machine setup parameters into the tags defined for the **PLC1** unit by running the following script:

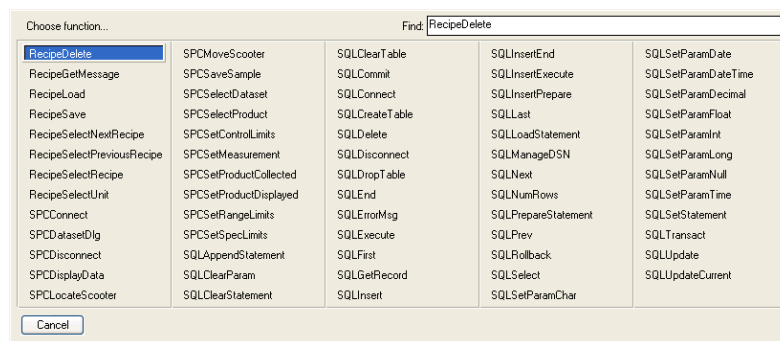
```
RecipeLoad("c:\recipe\machine.csv", "PLC1", Setup);
```

MACHINE.CSV							
	1	2	3	4	5	6	7
1	:Item Name	Item Type	Unit	Recipe	Recipe	Recipe	
2	:Names		PLC1	Setup1A	Setup2A	Setup3A	
3	PARM1	Analog	PARM1	11	21	31	
4	PARM2	Analog	PARM2	12	22	99	
5	PARM3	Analog	PARM3	13	23	66	
6	PARM4	Analog	PARM4	14	24	34	
7	PARM5	Analog	PARM5	11	21	31	
8	PARM6	Analog	PARM6	12	22	32	
9	PARM7	Analog	PARM7	13	23	33	
10	PARM8	Analog	PARM8	14	24	34	
11	PARM9	Analog	PARM9	15	25	35	
12							

Using Recipes in InTouch

You use InTouch QuickScripts to interact with your recipe template files. Recipe Manager includes a set of script functions that can be inserted into QuickScripts. Using scripts containing these functions, you can select, modify, insert, or delete records in your recipe template file.

You add recipe functions to any type of script using the InTouch script editor. The following figure shows the InTouch script editor's window listing recipe functions. All recipe functions are identified by the prefix **Recipe** as part of the function name.



InTouch recipe functions read and write directly to the recipe template file. Therefore, the Recipe Manager program does not need to be running in order for the recipe functions to run properly in InTouch QuickScripts.

If the recipe template file is being used by the InTouch HMI, any new recipes you create or any changes you make to existing recipes cannot be written to the recipe template file. Recipe Manager only creates recipe template files. After you create Recipe template files, close Recipe Manager.

To automatically insert a recipe function into a script

- 1 Open an InTouch script editor.
- 2 Place the cursor within the script where you want to insert a recipe function.
- 3 Under **Functions**, click **Add-ons**. The **Choose function** dialog box appears.
- 4 Click the recipe function that you want to insert into your QuickScript. The dialog box closes and the function is inserted at the cursor position.

Loading and Saving Recipe Data From/to a Recipe File

Recipe Manager provides separate InTouch QuickScript functions to load and save recipe data within a recipe file.

RecipeLoad() Function

The RecipeLoad() function loads data from a recipe to a specific unit of tags in an InTouch application.

Category

Recipe

Syntax

```
RecipeLoad("Filename", "UnitName", "RecipeName");
```

Arguments

FileName

Name of the recipe template file. The value associated with *FileName* can be a string constant or a message tag containing the name of the recipe template file.

UnitName

Name of the specific unit in the designated recipe template file. The RecipeSelectUnit() function returns a value to this parameter. The value associated with *UnitName* can be a string constant or a message tag that contains the name of the unit.

RecipeName

Name of the specific recipe in the designated recipe template file. The value associated with *RecipeName* can be a string constant or a message tag that contains the name of the recipe.

Example

The following statement loads the values of the Recipe1 recipe to the tags defined for Unit:

```
RecipeLoad("c:\recipe\recfile.csv", "Unit1",  
  "Recipe1");
```

RecipeSave() Function

The RecipeSave() function saves a new recipe or changes made to an existing recipe to a specified recipe template file.

Category

Recipe

Syntax

```
RecipeSave("Filename", "UnitName", "RecipeName");
```

Arguments*FileName*

Name of the recipe template file. The value associated with *FileName* can be a string constant or a message tag containing the name of the recipe template file.

UnitName

Name of the specific unit in the designated recipe template file that will be used by the function. The RecipeSelectUnit() function returns a value to this parameter. The value associated with *UnitName* can be a string constant or a message tag that contains the name of the unit.

RecipeName

Name of the specific recipe in the designated recipe template file. The value associated with *RecipeName* can be a string constant or a message tag that contains the name of the recipe.

Example

The following example saves changes made to the Recipe3 recipe in the recfile.csv file. If Recipe3 does not currently exist in the recfile.csv file, it is created. The values set the values of the tags defined for Unit2:

```
RecipeSave("c:\recipe\recfile.csv", "Unit2",  
  "Recipe3");
```

Deleting Recipes From a Recipe File

Use the RecipeDelete function to delete a recipe from a specified recipe template file.

RecipeDelete() Function

The RecipeDelete function deletes a recipe from a specified recipe template file.

Category

Recipe

Syntax

```
RecipeDelete ("Filename", "RecipeName");
```

Arguments

FileName

Name of the recipe template file. The value associated with *FileName* can be a string constant or a message tag containing the name of the recipe template file.

RecipeName

Name of the specific recipe in the designated recipe template file. The value associated with *RecipeName* can be a string constant or a message tag that contains the name of the recipe.

Example

The following statement deletes the Distlt1 recipe from the recfile.csv file:

```
RecipeDelete ("c:\recipe\recfile.csv", "Distlt1");
```

Selecting Units (Tag Ingredient Mappings)

Use the `RecipeSelectUnit()` function to select the unit of tags to which the current recipe values are loaded.

RecipeSelectUnit() Function

The `RecipeSelectUnit()` function opens the **Select a Unit** dialog box so that the run-time user can select a unit. The selected unit name is returned to a message tag. Both the `RecipeSelectRecipe()` and `RecipeSelectUnit()` functions are used in conjunction with the `RecipeLoad()` function.

Category

Recipe

Syntax

```
RecipeSelectUnit ("Filename", UnitName, Number);
```

Arguments

FileName

Name of the recipe template file. The `FileName` argument can be a string constant or a message tag containing the name of the recipe template file.

UnitName

Message tag to which the name of the selected unit is written. Actual message tag without quotes or a string literal.

Number

Maximum string length returned to the argument. In InTouch, string (message) tags have a maximum length of 131 characters. Use 131 for this argument unless you have reduced the maximum string length of the InTouch tag. Number or integer tag.

Example

The following statement causes the **Select a Unit** dialog box to open:

```
RecipeSelectUnit ("c:\recipe\recfile.csv",  
    UnitName, 131);
```

After a Unit is selected, its name is returned to the `UnitName` tag.

Selecting Individual Recipes from a Recipe File

Recipe Manager includes a set of functions to select an individual recipe from the recipe file. When you use these functions in a script, you can select a recipe from the file by its name or the next or previous recipe in sequence within the file.

RecipeSelectRecipe() Function

The `RecipeSelectRecipe()` function opens the **Select a Recipe** dialog box so that the run-time user can select a recipe. The selected recipe name is returned to a message tag.

Category

Recipe

Syntax

```
RecipeSelectRecipe ("FileName", RecipeName,  
    Number);
```

Arguments

FileName

Name of the recipe template file. The *FileName* argument can be a string constant or a message tag containing the name of the recipe template file.

RecipeName

Message tag to which the name of the selected recipe is written. Actual message tag without quotation marks or a string literal.

Number

Maximum string length returned to the argument. InTouch message tags have a maximum length of 131 characters. Use 131 for this parameter unless you have reduced the maximum string length of the InTouch tag. Number or Integer tag.

Example

The following statement causes the **Select a Recipe** dialog box to open:

```
RecipeSelectRecipe ("c:\recipe\recfile.csv",  
    RecipeName, 131);
```

After a recipe is selected from the dialog box, its name is returned to the `RecipeName` tag.

RecipeSelectNextRecipe() Function

The `RecipeSelectNextRecipe()` function selects the next recipe in the recipe template file.

Category

Recipe

Syntax

```
RecipeSelectNextRecipe("Filename", RecipeName,  
    Number);
```

Arguments

FileName

Name of the recipe template file. The *FileName* argument can be a string constant or a message tag containing the name of the recipe template file.

RecipeName

Message tag that contains the recipe name to use as a starting point (before the function is executed) and the selected recipe name (after the function is executed). Actual message tag without quotation marks or a string literal.

Number

Maximum string length returned to the argument. In InTouch, string (message) tags have a maximum length of 131 characters. Use 131 for this parameter unless you have reduced the maximum string length of the InTouch tag. Number or integer tag.

Example

The following statement reads the current value of the tag *RecipeName* and returns the next recipe on file. If the value of *RecipeName* is blank or cannot be found, the first recipe on file is returned. If *RecipeName* currently contains the last Recipe Name on file, it is returned unchanged. Recipes are saved in the recipe template file in the order in which they are created.

```
RecipeSelectNextRecipe("c:\recipe\recfile.csv",  
    RecipeName, 131);
```


RecipeSelectPreviousRecipe() Function

The RecipeSelectPreviousRecipe() function selects the previous recipe defined in the recipe template file.

Category

Recipe

Syntax

```
RecipeSelectPreviousRecipe ("FileName", RecipeName,  
    Number);
```

Arguments

FileName

Name of the recipe template file. The *FileName* argument can be a string constant or a message tag containing the name of the recipe template file.

RecipeName

Message tag that contains the recipe name to use as a starting point (before the function is executed) and the selected recipe name (after the function is executed). Actual message tag without quotation marks or a string literal.

Number

Maximum string length returned to the parameter. In InTouch, Message tags have a maximum length of 131 characters. Use 131 for this parameter unless you have reduced the maximum string length of the InTouch tag. Number or Integer tag.

Example

The following statement causes the system to read the current value of the tag *RecipeName* and return the previous Recipe Name on file. This returned string will be stored in *RecipeName* and will overwrite the current value. If the value of *RecipeName* is blank or cannot be found, the last recipe on file is returned. If *RecipeName* currently contains the first Recipe Name on file, it is returned unchanged. (Recipes are saved in the order in which they are created.)

```
RecipeSelectPreviousRecipe ("c:\recipe\recfile.csv",  
    RecipeName, 131);
```

Understanding Error Messages Returned by Recipe Script Functions

You troubleshoot recipe applications using diagnostic error codes returned by a recipe function. This section includes a list of recipe function error codes and how to use the `RecipeGetMessage()` function to show the message associated with an error code.

The `RecipeLoad()` function sets the value of the analog `ErrorCode` tag to 0 if it is successful. If `RecipeLoad()` fails, it sets the `ErrorCode` tag to the number of the specific error condition.

To retrieve the error code of a recipe function, it must be equated to an InTouch analog tag. The following example shows a script statement to return a recipe function error code:

```
ErrorCode = RecipeLoad(FileName, UnitName, RecipeName);
```

Displaying Error Code Messages

Each recipe function returns a number that represents the error condition for the function. By using the `RecipeGetMessage()` function in an InTouch Data Change script, the corresponding error code can be written to an analog tag and the associated error code message can be written to a message tag.

The following code example shows a Data Change script.

```
RecipeGetMessage(ErrorCode, ErrorMessage, 131);
```

This script runs automatically whenever the value of the `ErrorCode` tag changes. When this script runs, the `RecipeGetMessage()` function reads the current numeric value of the `ErrorCode` tag and returns the message associated with that value to the `ErrorMessage` tag.

The following table lists possible error codes, their corresponding error messages, and descriptions:

Value	Error Message	Description
0	Success	The called recipe function executed successfully.
-1	No Such Recipe Template	The specified recipe template file does not exist.
-2	View Not Active	The recipe function called by another program cannot execute because WindowViewer is not running.

Value	Error Message	Description
-3	Out of Memory	There is not enough memory to complete the current activity.
-4	Line too long in recipe template file	A line in the recipe template file exceeds the maximum allowed length.
-5	Truncated line in the recipe file	A line in the recipe template file is truncated.
-6	Not a valid recipe template file	The specified file is not a valid recipe template file.
-7	Expecting "unit" or "recipe"	A unit name or recipe name is missing from the recipe template file.
-8	No units defined in recipe template file	No units have been defined in the Units Definition template of the recipe file.
-9	Recipe name not found in recipe template file	The specified recipe is not defined in the recipe template file.
-10	Unit name not found in recipe template file	The specified unit name is not defined in the unit definition template file
-12	Expecting "Analog", "Discrete", "Message"	An incorrect type has been entered for an item in the recipe template file. Valid types are analog, discrete or message.
-13	Type of tag mismatches "Analog", "Discrete", "Message"	The specified tag does not match the item type. For example, a recipe item is defined as analog and a message tag has been defined as the unit for it.
-14	Invalid discrete value, expecting "0", "1"	An incorrect value has been entered for a discrete tag in the recipe template file. The only valid values for discrete tags are 0 or 1.
-15	Unable to open temporary file	The temporary file cannot be opened possibly because of insufficient free disk space.

Value	Error Message	Description
-16	Write error while saving recipe template file	An error occurred while saving the recipe template file.
-17	User did not select	The user selected Cancel in the Select a Recipe dialog box instead of a recipe name.
-19	Recipe template in use by another application	The recipe template file specified is open and, therefore, cannot be accessed by WindowViewer.

RecipeGetMessage() Function

The `RecipeGetMessage()` function takes an error number (returned by some other recipe function) and returns the plain text error message for that error number.

Category

Recipe

Syntax

```
RecipeGetMessage (Analog_Tag, Message_Tag, Number) ;
```

Arguments

Analog_Tag

Error number for which you want to get the error message.

Message_Tag

Actual message tag with no quotes or string literal.

Number

The *Number* argument sets the maximum length of the string returned with the *Message_Tag* argument. By default, InTouch message tags are set to the maximum length of 131 characters. Use 131 for this parameter unless you have reduced the maximum string length of *Message_Tag* in the InTouch Tagname Dictionary. The *Number* argument can be a constant or an integer tag containing a number.

Example

By using the `RecipeGetMessage()` function in an InTouch Data Change QuickScript, the error code is written to the `ErrorCode` tag and the associated error code message is written to the `ErrorMessage` tag:

```
Data Change Script Tagname[.field]:ErrorCode
Script body:RecipeGetMessage(ErrorCode,
    ErrorMessage,131);
```

This QuickScript automatically runs whenever the value of the `ErrorCode` tag changes. When this QuickScript runs, the `RecipeGetMessage()` function reads the current numeric value of the `ErrorCode` tag and returns the message associated with that value to `ErrorMessageTag`.

```
ErrorCode = RecipeLoad
("c:\App\recipe.csv", "Unit1", "cookies");
RecipeGetMessage(ErrorCode, ErrorMessageTag,
    131);
```

Applying Security to Recipes

Access to recipes can be controlled by defining an Item Name in the recipe template file that sets the minimum security access level required to load, save, or delete a recipe.

In the following file sample, the `SecurityLevel` Item Name is defined as an analog tag. The `Review` unit contains the `SecurityLevel` analog tag for this item. Each recipe defines a value that is loaded into the `SecurityLevel` tag when the recipe is loaded into the `Review` unit.

MACHINE.CSV							
1	2	3	4	5	6	7	
:Item Name	Item Type	Unit	Unit	Recipe	Recipe	Recipe	
2	:Names	Review	PLC1	Setup1A	Setup2A	Setup3A	
3	PARM1	Analog		PARM1	11	21	31
4	PARM2	Analog		PARM2	12	22	99
5	PARM3	Analog		PARM3	13	23	66
6	PARM4	Analog		PARM4	14	24	34
7	PARM5	Analog		PARM5	11	21	31
8	PARM6	Analog		PARM6	12	22	32
9	PARM7	Analog		PARM7	13	23	33
10	PARM8	Analog		PARM8	14	24	34
11	PARM9	Analog		PARM9	15	25	35
12	SecurityLevel	Analog	SecurityLevel		2000	5000	7000

You can create a window containing an access denied message that is shown whenever your security access level is invalid for a selected recipe. To do so, the selected recipe must be loaded into a unit that contains only an analog tag to which the selected recipe's security level value is loaded for verification.

For example:

```
RecipeSelectRecipe("c:\recipe\machine.csv",  
    MyRecipe, "131");
```

The **Select a Recipe** dialog box appears. After you select a Recipe Name, it is returned to the RecipeName tag and the script continues running.

```
RecipeLoad( "c:\Recipe\Machine.csv", "Review", MyRecipe  
    );  
IF SecurityLevel <= $AccessLevel THEN  
    Status =RecipeLoad( "c:\Recipe\Machine.csv",  
        "PLC1", MyRecipe );  
    ELSE Show "Access Denied";  
ENDIF;
```

When this script runs, if your access level is equal to or greater than 7000, the selected recipe's values are loaded into PLC1 unit's tags. If not, the **Access Denied** window appears and the recipe is not loaded into PLC1.

Chapter 3

Working with SQL Databases from InTouch

A database stores information in tables that share a common attribute or field. Structured Query Language (SQL) is the language you use to access that information in the form of a query. SQL Access Manager allows you to access, modify, create, and delete database tables with queries.

SQL Access Manager is an optional program that can be installed with InTouch. SQL Access Manager allows you to:

- Create and run complex queries. These queries can be built dynamically or saved in external files. Additionally, these queries can contain parameters that are passed to the query at run time.
- Run SQL statements supported by your database and retrieve the results from the query. You can also use stored procedures with SQL Access Manager, although not all stored procedure functions are fully supported.

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

You can use SQL Access Manager to transfer data, such as batch recipes from a SQL database to an InTouch application. SQL Access Manager can also be used to transfer run-time data, alarm status, or historical data from InTouch to a database. For example, after a machine cycle is completed, a company wants 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.

SQL Access Manager consists of a program and a set of SQL functions. The SQL Access Manager program creates and associates database columns with InTouch tags. The process of associating database columns and InTouch database tags is called binding. Binding the InTouch database tags to database columns allows SQL Access Manager to directly manipulate InTouch data stored in a database.

SQL Access Manager saves the database field names and their associations in a comma-separated variable file named SQL.DEF. This file resides in the InTouch application folder and can be viewed or modified using SQL Access Manager or any text editor, such as Notepad. SQL Access Manager also creates Table Templates that define the structure and format of the database used with InTouch.

SQL functions can be used in scripts to automatically run based on operator input, a tag value changing, or when a particular set of conditions exist. These functions allow you to select, modify, insert, or delete records in the tables you choose to access. For example, if an alarm condition exists, the application can run a script that includes the `SQLInsert()` or `SQLUpdate()` functions that save all of the applicable data points and the state of the alarm.

Setting Up a Data Source

SQL Access Manager is an ODBC-compliant application that communicates with any database that supports an available ODBC driver or an OLE DB provider.

You can configure the connection string to the database by several methods:

- Use the Microsoft ODBC Administrator program to configure the ODBC driver for use with SQL Access Manager.
- Run the SQLConnect() function and specify an OLE DB provider with argument values. For more information, see SQL Server Database Applications on page 128.
- Set the database connection string with an external UDL file.

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 appears.

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

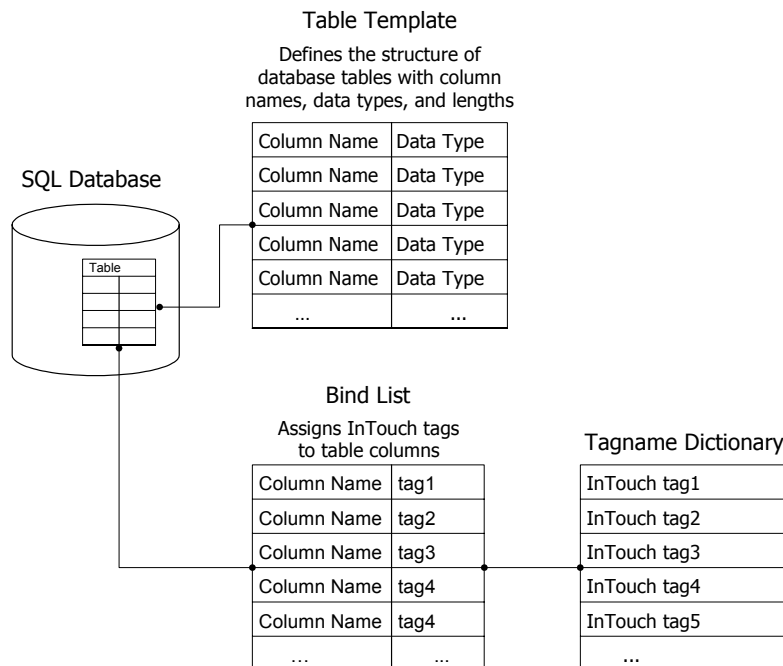
- 3 Click **OK**.

Note When you create an ODBC Data Source, an ODBC.INI file is created in the Windows directory. You can manually edit the ODBC.INI file.

Mapping InTouch Tags to Database Columns

You can map InTouch Tags to database columns. This is done with a Bind List. Most SQL Access functions use the Bind List to enable InTouch tags to access data in SQL database tables.

A Bind List associates database table columns to tags in the InTouch Tagname Dictionary. A Bind List also includes a Table Template that describes the format of the database tables.

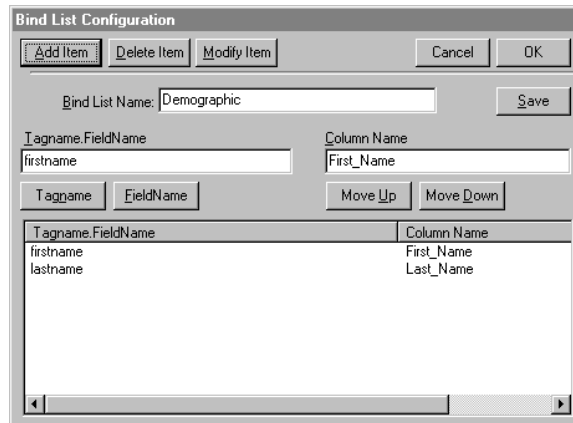


When you run a script containing the `SQLInsert()`, `SQLSelect()` or `SQLUpdate()` functions, the Bind List is updated to specify which InTouch tags are used and which table columns are associated with these tags.

To create a new Bind List

- 1 On the **Special** menu, point to **SQL Access Manager**, and then click **Bind List**. A message requests confirmation to create the `SQL.DEF` file.
- 2 Click **Yes** to create the `SQL.DEF` file. The **Select a Bind List** dialog box appears.

- 3 Click **New**. The **Bind List Configuration** dialog box appears.



- 4 In the **Bind List Name** box, type the Bind List Name. A Bind List name can be up to 32 characters.
- 5 To define the tags for the Bind List, do one of the following:
- In the **Tagname.FieldName** box, type an InTouch tag name. You can also add an optional tag dotfield in the form *tag_name.dotfield_name*.
 - Double-click **Tagname** to select an existing tag. The **Select Tag** dialog box appears. Select a tag from the list.

Note I/O type tags that are not used in your application, but are specified in a SQL Access Bind List, are activated (advised to the DAServer) as soon as WindowViewer starts.

- 6 Select the dotfield to append to the tag by one of the following:
- In the **Tagname.FieldName** box, type a period and the dotfield name after the tag name
 - Click **FieldName**. The **Choose field name** dialog box appears. Click the dotfield that you want to append to the tag.
- 7 In the **Column Name** box, type the name of the column.

A column name can be up to 30 characters in length. 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= "[Valve ID] = " + text
(tagname, "#");
```

- 8 Position the tag within the Bind List by doing one of the following:
 - Click **Move Up** to move the selected tag up one level in the list.
 - Click **Move Down** to move the selected tag down one level in the list.
- 9 Click **Add Item** to add your new **Tagname.FieldName** and **Column Name** to the Bind List.
- 10 Click **OK** to save your new Bind List configuration and close the dialog box.

Configuring the SQL Server String Delimiter in Bind Lists

The `SQLInsert()` and the `SQLUpdate()` functions use a default format that encloses message strings within single quotation marks. Some SQL databases expect to receive message strings enclosed by another type of delimiter. For example, Oracle 8.0 expects to receive a date string surrounded by brackets. When this occurs, the `Delim()` function must be used.

In the **Bind List Configuration** dialog box **Column Name** field, after the column name, use the `delim()` function. The keyword "delim" must be entered followed by:

- a left parenthesis
- the left delimiter
- the list separator character defined in the system's regional settings
- the right delimiter
- a right parenthesis

Example for an English system: `datestring delim (','')`

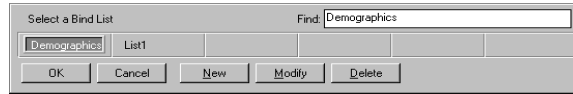
Example for a German system: `datestring delim (';')`

To use the same delimiter for both left and right, specify the delimiter in parentheses without the separator, as shown in the following example:

```
datestring delim ('')
```

To modify a Bind List

- 1 On the **Special** menu, point to SQL Access Manager, and then click **Bind List**. The **Select a Bind List** dialog box appears.



- 2 Select the Bind List name that you want to change, and then click **Modify**. The **Bind List Configuration** dialog box appears.
- 3 Modify the required items.
- 4 Click **OK** to save your changes and close the dialog box.

To modify a Bind List with Excel

SQL Access Manager saves the configuration information for the Bind Lists and table templates to the SQL.DEF file. This file is formatted as a Comma Separated Value (CSV) file.

The SQL.DEF file can be modified with any product that supports Comma Separated Value files like Excel.

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
```

To delete a Bind List

- 1 On the **Special** menu, point to SQL Access Manager, and then click **Bind List**. The **Select a Bind List** dialog box appears.
- 2 Select the Bind List name that you want to delete.
- 3 Click **Delete**. A message appears requesting confirmation to delete the Bind List.
- 4 Click **Yes** to delete the selected Bind List. The **Bind List Configuration** dialog box reappears.
- 5 Click **OK** to close the dialog box.

Defining the Structure of a New Table

A Table Template defines the structure and format for new tables you create in the database. The Table Template is stored in the SQL.DEF file.

To create a new Table Template

- 1 On the **Special** menu, point to SQL Access Manager, and then click **Table Template**.
- 2 Click **New**. The **Table Template Configuration** dialog box appears.

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

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

A Table Template name can be up to 32 characters without an index. If you are creating an index, unique or otherwise, the Table Template name must not exceed 24 characters.

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

A column name can be up to 30 characters.

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

- 6 In the **Index Type** area, select one of the following options:
 - **Unique:** Each column value must be unique.
 - **Non-Unique:** Each column value is not required to be unique.
 - **None:** No index.

Note When you run a script containing the `SQLCreateTable()` function, an index file is automatically created.

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

Note InTouch does not support null data.

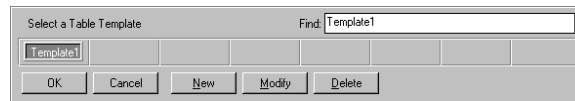
When inserting data, if a value has not been entered for a tag, null values are assigned by the type of tag.

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

- 8 Click **Add Item** to add your new column name, column type, length, and index type to the Table Template.
- 9 Click **OK** to save your new Table Template configuration and close the dialog box.

To modify a Table Template

- 1 On the **Special** menu, point to SQL Access Manager, and then click **Table Template**. The **Select a Table Template** dialog box appears.



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

To delete a Table Template

- 1 On the **Special** menu, point to SQL Access Manager, and then click **Table Template**. The **Select a Table Template** dialog box appears.
- 2 Select the Table Template name that you want to delete.
- 3 Click **Delete**. A message appears requesting confirmation to delete the Table Template.
- 4 Click **Yes**. The **Table Template Configuration** dialog box reappears.
- 5 Click **OK** to close the dialog box.

Working with Database Applications

SQL Access Manager supports Oracle, Microsoft SQL Server, and Microsoft Access databases. Each database's requirements are unique. This section includes separate sections that describe how to configure the connection between each database and SQL Access Manager.

SQL Server Database Applications

You use the `SQLConnect()` function in an InTouch QuickScript to connect to a Microsoft SQL Server database. The `SQLConnect()` function logs on a user to a SQL Server database and opens a connection. The connection string used by the `SQLConnect()` function is formatted as follows:

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

The *ConnectionID* argument is an integer tag containing a session number. This session number is used by almost every other SQL Access function to reference the connection to the SQL Server database. The session number increments by 1 with each `SQLConnect()` function call.

The following table describes the `SQLConnect()` function attributes used by Microsoft SQL Server:

Attribute	Value
Provider	SQLOLEDB
Data Source	Server name where the database is installed
Initial Catalog	Database name
User ID	Logon ID, case sensitive
Password	Password, case sensitive

```
"Provider=SQLOLEDB.1;User ID=UserIDStr;
Password>PasswordStr;Initial
Catalog=DatabaseName;Data Source=ServerName;"
```


SQL Access Manager associates the four types of InTouch tags (discrete, integer, real, and message) with other SQL Server database data types.

Data Type	Length	Range	Tag Type
char	8,000 characters	1 to 131	Message
int		-2,147,483,648 to 2,147,483,647	Integer
float	15 digits	-1.79E ⁺³⁰⁸ to 1.79E ⁺³⁰⁸	Real

The char data type contains fixed-length character strings. InTouch message tags 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 tags are limited to 131 characters. If a message tag value contains more characters than the length specified for a database field, the char string is truncated when inserted into the database.

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

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

Microsoft Access Database Applications

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

The `SQLConnect()` function is used to connect to Microsoft Access databases. Running a script containing the `SQLConnect()` function logs you on to the database server and opens a connection to allow other SQL functions to be run. The connection string used by `SQLConnect()` is formatted as follows:

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

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

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

The valid data types that SQL Access Manager supports depends on the version of the ODBC driver being used.

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

The text data type contains fixed length character strings and are used with InTouch Message tags. A length must be specified. Microsoft Access databases support text fields with a maximum length of 255 characters. InTouch Message tags 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.

Oracle Database Applications

To establish communication between SQL Access and an Oracle database, you must connect to it by running a script containing the `SQLConnect()` function.

To communicate with an Oracle 8.0 database

- 1 Verify the Oracle OLEDB Provider (MSDAORA.DLL) file is installed on the computer running InTouch. 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.

The connection string used by the `SQLConnect()` function is formatted as follows:

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

The following table describes the function attributes used by Oracle:

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

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

The following table lists the valid data types that SQL Access Manager supports for an Oracle database.

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

To log the date and time to an Oracle 8.0 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.
- 2 In the **Tagname.FieldName** box, type the tag that you want to use. For example, DATE_TIME_TAG.
- 3 In the **Column Name** box, type the name of the Oracle date field. If you are using Oracle 8.0, use the `delim()` function to specify any delimiters. The `delim()` function is not required if you are using Oracle 9.2 or later.
- 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')";
```

After the QuickScript runs in WindowViewer, the date appears in the following format:

```
TO_DATE('08/22/06 23:32:18' , 'mm/dd/yy
hh24:mi:ss')
```

Performing Common SQL Operations in InTouch

InTouch uses SQL Access functions to interact with information stored in a database. These SQL Access functions enable you to write scripts that select, modify, insert, or delete database records.

SQL actions are synchronous. When you run a database QuickScript from an InTouch application, control does not return to InTouch until the database action requested by the function is complete.

SQL Access functions adhere to punctuation standards that describe the type of arguments associated with a function. When an argument is entered in a script string surrounded by quotation marks ("Arg1") that exact string is used. If no quotation marks are used, the argument value is assumed to be a tag name and the current value of the tag is associated with the argument.

Most SQL functions return a result code. If the result code is non-zero, the function failed and other actions should be taken. The result code can be used by the `SQLErrorMsg()` function.

You insert SQL functions in your QuickScripts using the InTouch QuickScript editor. The general procedure to insert a SQL function into a script includes the following steps:

To add a SQL function to a script

- 1 Start InTouch WindowMaker.
- 2 Open the QuickScript with the QuickScript editor.
- 3 Place the cursor in the script where you want to insert the SQL function.
- 4 In the **Functions** area, click **Add-ons** to show the **Choose function** dialog box.
- 5 Click on the SQL function that you want to insert into the QuickScript. The script updates and shows the SQL function that you inserted.

The arguments associated with SQL Access functions consist of the following:

- *BindList*
Corresponds to a Bind List name defined in the SQL.DEF file.
- *ConnectionID*
The *ConnectionID* argument refers to the name of a memory integer tag that holds the number (ID) assigned by the `SQLConnect()` function to each database connection.

- *ConnectionString*

The *ConnectionString* identifies the database system and any additional logon information. It is entered in the following format:

```
"DSN=data source name[;attribute=value
[;attribute=value]...]"
```

Microsoft SQL Server Connection Strings

- Microsoft OLE DB Provider for SQL Server (recommended use).

```
"Provider=SQLOLEDB.1;User ID=sa;
Password=;Initial Catalog=MyDB;Data
Source=MyServer;"
```

The OLE DB Provider for SQL Server is sqloledb.

- Microsoft OLE DB Provider for SQL Server (recommended use)

```
"Provider=SQLOLEDB.1;uid=sa;pwd=;Database=MyDB"
```

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

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

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

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

Oracle Connection Strings

- Microsoft OLE DB Provider for Oracle (recommended use)

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

Microsoft Access Connection Strings

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

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

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

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

- *ErrorMsg*

Message variable containing a text description of the error.

- *FileName*

Name of the file in which the information is contained.

- *MaxLen*

Maximum size of the column associated with a parameter. This argument determines whether the data 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 data is varying character type. If greater, then the data is long varying character type.

- *OrderByExpression*

Defines the columns and either ascending or descending sort order. 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 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 are:

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
Time	4-byte date time value
No Type	No data type

- *ParameterValue*

Actual value to set.

- *Precision*

Is the decimal value's precision, the maximum 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.

- *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 *TableName* parameter contains the name of the table you want to access or create in the database.

- *TemplateName*

The *TemplateName* parameter is the name of the template in the SQL.DEF file that defines the table.

- *WhereExpr*

Defines a condition that can be either true or false for any row of the table. The function 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 enclosed within single quotation marks.

The following example selects all rows whose Name column contains the value EmployeeID:

```
Name='EmployeeID'
```

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

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

The following example selects all records whose temperature column contains a value greater than 350:

```
temperature>350
```

Connecting and Disconnecting the Database

Use the `SQLConnect()` and `SQLDisconnect()` functions in a script to connect to and disconnect from a SQL database.

SQLConnect() Function

You use the `SQLConnect()` function in an InTouch QuickScript to connect to the database specified by the *ConnectString* argument.<

`SQLConnect()` returns a value to the *ConnectionID* argument that is used as a parameter in all subsequent SQL functions. You must have a Bind List defined in the application folder before using the `SQLConnect` function in a script.

Category

SQL

Syntax

```
[ResultCode=] SQLConnect (ConnectionID,
    "ConnectString");
```

Arguments

ConnectionID

Name of a memory integer tag that holds 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()` function.

Remarks

You must have a Bind List (a `SQL.DEF` file) in the application folder. This function does not work without it.

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

Examples

The following statements connects to IBM OS/2 Database Manager and to the database named `SAMPLE`:

```
[ResultCode=] SQLConnect (ConnectionID, "DSN=OS2DM;
    DB=SAMPLE");
```

This function returns a value to the *ConnectionID* variable that is used as a parameter in all subsequent SQL Functions.

```
"DSN=data source name[;attribute=value
    [;attribute=value]..."
```

SQLDisconnect() Function

The SQLDisconnect() function disconnects you from the database and cleans up all unreleased resources that were obtained for SQLPrepareStatement() and SQLInsertPrepare() functions.

Category

SQL

Syntax

```
[ResultCode=]SQLDisconnect (ConnectionID);
```

Argument*ConnectionId*

Name of a memory integer tag that holds the number (ID) assigned by the SQLConnect() function to each database connection.

See Also

SQLConnect()

Creating a New Table

You use the SQLCreateTable() function in an InTouch QuickScript to create a table in the database using the parameters from a specified Table Template.

SQLCreateTable() Function

You use the SQLCreateTable() function in an InTouch QuickScript to create a table in the database using the parameters from a specified Table Template. Table Templates are defined in the SQL.DEF file, which includes the structure of a database table.

Category

SQL

Syntax

```
[ResultCode=]SQLCreateTable (ConnectionID,  
    TableName, TemplateName);
```

Arguments*ConnectionID*

Name of a memory integer tag that holds the number (ID) assigned by the SQLConnect() function to each database connection.

TableName

Name of the database table you want to create.

TemplateName

Name of the template definition you want to use.

Examples

The following example of the `SQLCreatTable()` function creates a table named *BATCH1* with the column names and data types defined in the `OutputVal` template:

```
ResultCode=SQLCreateTable (ConnectionID, "BATCH1",  
"OutputVal");
```

See Also

`SQLConnect()`

Deleting a Table

You use the `SQLDropTable()` function in an InTouch QuickScript to drop a table from the database.

SQLDropTable() Function

You use the `SQLDropTable()` function in an InTouch QuickScript to drop a table from the database. After the QuickScript containing the `SQLDropTable()` function finishes, the table is no longer recognized and does not respond to any SQL statements.

Category

SQL

Syntax

```
[ResultCode=]SQLDropTable (ConnectionID,  
TableName);
```

Arguments*ConnectionID*

Name of a memory integer tag that holds the number (ID) assigned by the `SQLConnect()` function to each database connection.

TableName

Name of the table that you want to drop from the database.

Example

The following example of the `SQLDropTable()` function drops the *BATCH1* table from the database:

```
ResultCode=SQLDropTable (ConnectionID, "BATCH1");
```

See Also

`SQLConnect()`

Retrieving Data from a Table

You can use a set of SQL functions in scripts to retrieve data from a database and write the values to InTouch tags.

- The `SQLSelect()` function retrieves information from a table and places this information in the form of records into a temporary Results Table created in memory.
- The `SQLGetRecord()` function retrieves the record specified by *RecordNumber* from the current selection buffer.
- The `SQLNumRows()` function returns the number of table rows that met the criteria specified in a previous `SQLSelect()` function.
- The `SQLFirst()` function retrieves the first record of the Results Table created by the last `SQLSelect()` function.
- The `SQLNext()` function retrieves the next record of the Results Table created by the last `SQLSelect()` function.
- The `SQLPrev()` function retrieves data from the previous row of the logical table and fetch values from that row into InTouch tags.
- The `SQLLast()` function retrieves the last row of the logical table and fetch values from that row into InTouch tags.
- The `SQLEnd()` function frees memory that stores the contents of the Results Table associated with `ConnectionId`.

The `SQLFirst()`, `SQLPrev()`, `SQLNext()`, `SQLLast()`, and `SQLGetRecord()` functions retrieve data from specified rows of the logical table and save it as InTouch tag values. If a field is `NULL`, the value of the associated InTouch tag is set to 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 from the database to the associated InTouch message tag.

SQLSelect() Function

The SQLSelect() function retrieves records from a table. When the script containing the SQLSelect() function is processed, the retrieved records are placed in a temporary Results Table in memory. These records can be browsed using the SQLFirst(), SQLLast(), SQLNext() and SQLPrev() functions.

Important Always call the SQLEnd() function after the script containing the SQLSelect() function ends to free memory used by the Results Table.

Category

SQL

Syntax

```
[ResultCode=] SQLSelect (ConnectionID, TableName,  
    BindList, WhereExpr, OrderByExpression);
```

Arguments

ConnectionID

Name of a memory integer tag that holds the number (ID) assigned by the SQLConnect() function to each database connection.

TableName

Name of the database table to access.

BindList

Defines which InTouch tags are used and which database.

WhereExpr

Defines a condition that can be either true or false for any row of the table. The SQLSelect() function extracts data from only those rows in which the *WhereExpr* condition is true. The expression must be in the following format:

ColumnName *comparison_operator* expression.

Note If the comparison is made with a character expression, the expression must be enclosed within single quotes.

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

```
name='EmployeeID'
```

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

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

The following example selects all rows whose temperature column contains a value greater than 350:

```
temperature>350
```

WhereExpr - Memory message Tag

```
OrderByExpr - Memory message Tag
Speed_Input - Memory Real - User Input Analog
Serial_Input - Memory Message - User Input
String
```

Analog Example

```
WhereExpr = "Speed = " + text
(Speed_Input, "#.##");
```

Because Speed_Input is a number, it must be converted to text so it can be concatenated to the WhereExpr string.

String Example

```
WhereExpr = "Ser_No = \" +
Serial_input + "\"";
```

Because Serial_Input is a string it must have single quotes around the value for example:WhereExpr = "Ser_No='125gh'";

String Example using the like statement

```
WhereExpr = "Ser_No like \"-\""
```

When using the Like comparison operator the % char can be used as a wild card.

String and Analog Example using a Boolean AND operator

```
WhereExpr = "Ser_No = \" + Serial_input + "\" +
\" and \" + \"Speed = \" +
text(Speed_Input, "#.##"); OrderByExpr = "";
```

If the order does not matter, use a null string as shown above.

SQLSelect using WhereExpr tag

```
ResultCode = SQLSelect(Connect_Id,TableName,  
BindList,  
WhereExpr,OrderByExpr);  
Error_msg = SQLErrorMsg( ResultCode );
```

SQLSelect WhereExpr built in function

```
ResultCode = SQLSelect(Connect_Id,TableName,  
BindList,  
"Ser_No = \'' + Serial_input + '\'",  
OrderByExpr);  
Error_msg = SQLErrorMsg( ResultCode );
```

OrderByExpr

Defines the direction to sort data within a table column.

Only column names can be used to sort and the expression must be in this form:

ColumnName [ASC|DESC]

The following example sorts a table in ascending order by the data from the manager column:

```
"manager ASC"
```

You can also sort by multi-columns where the expression is in the form:

ColumnName [ASC|DESC],

ColumnName [ASC|DESC]

The next example sorts the selected table by the temperature column in ascending order and the time column in descending order:

```
"temperature ASC,time DESC"
```


Examples

The following statement selects records from the BATCH table using a BindList named List1, whose column name type contains the value cookie. It will present the information sorted by the amount column in ascending order and the sugar column in descending order:

```
ResultCode=SQLSelect (ConnectionID, "BATCH",
    "List1", "type='cookie'", "amount ASC, sugar
    DESC");
```

The following statement selects all data in the database, do not specify a value for the *WhereExpr* and *OrderByExpr*:

```
ResultCode=SQLSelect (ConnectionID, "BATCH",
    "List1", "", "");
```

See Also

SQLFirst(), SQLConnect(), SQLLast(), SQLNext(), SQLPrev(), SQLEnd(), SQLSelect()

SQLGetRecord() Function

The SQLGetRecord() function retrieves the record specified by the *RecordNumber* argument from the current selection buffer.

Category

SQL

Syntax

```
[ResultCode=] SQLGetRecord (ConnectionID,
    RecordNumber);
```

Arguments*ConnectionID*

Name of a memory integer tag that holds the number (ID) assigned by the SQLConnect() function to each database connection.

RecordNumber

Actual record number to retrieve.

Example

```
ResultCode=SQLGetRecord (ConnectionID, 3);
```

See Also

SQLConnect()

SQLNumRows() Function

The SQLNumRows() function indicates how many rows met the criteria specified in the last SQLSelect() function. For example, if a *WhereExpr* argument is used to select all rows with a column name AGE, where AGE is equal to 45, the number of rows returned could be 40 or 4000. This may determine which function is processed next.

Category

SQL

Syntax

```
SQLNumRows (ConnectionID) ;
```

Argument*ConnectionID*

Name of a memory integer tag that holds the number (ID) assigned by the SQLConnect() function to each database connection.

Example

The following statement returns the number of rows selected to the NumRows integer tag:

```
NumRows=SQLNumRows (ConnectionID) ;
```

See Also

SQLConnect()

SQLFirst() Function

The SQLFirst() function selects the first record of the Results Table created by the last SQLSelect() function.

Category

SQL

Syntax

```
[ResultCode=] SQLFirst (ConnectionID) ;
```

Argument*ConnectionID*

Name of a memory integer tag that holds the number (ID) assigned by the SQLConnect() function to each database connection.

See Also

SQLConnect(), SQLSelect()

SQLNext() Function

The SQLNext() function selects the next record in sequence of the Results Table created by the last SQLSelect() function. A SQLSelect() function must be processed before running the SQLNext() function in a script.

Category

SQL

Syntax

```
[ResultCode=] SQLNext (ConnectionID) ;
```

Argument*ConnectionID*

Name of a memory integer tag that holds the number (ID) assigned by the SQLConnect() function to each database connection.

Example

```
ResultCode=SQLNext (ConnectionID) ;
```

See Also

SQLConnect(), SQLSelect()

SQLPrev() Function

The SQLPrev() function selects the previous record of the Results Table created by the last SQLSelect() function.

Category

SQL

Syntax

```
[ResultCode=] SQLPrev (ConnectionID) ;
```

Argument*ConnectionID*

Name of a memory integer tag that holds the number (ID) assigned by the SQLConnect() function to each database connection.

Remarks

A SQLSelect() function must be processed before using this command.

Example

```
ResultCode=SQLPrev (ConnectionID) ;
```

See Also

SQLConnect(), SQLSelect()

SQLLast() Function

The SQLLast() function selects the last record of the Results Table created by the previous SQLSelect() function.

Category

SQL

Syntax

```
[ResultCode=]SQLLast (ConnectionID) ;
```

Argument

ConnectionID

Name of a memory integer tag that holds the number (ID) assigned by the SQLConnect() function to each database connection.

Example

```
ResultCode=SQLLast (ConnectionID) ;
```

See Also

SQLConnect(), SQLSelect()

SQLEnd() Function

The SQLEnd() function is run after the SQLSelect() function to free memory used to store the contents of the Results Table.

Category

SQL

Syntax

```
[ResultCode=]SQLEnd (ConnectionID) ;
```

Argument

ConnectionID

Name of a memory integer tag that holds the number (ID) assigned by the SQLConnect() function to each database connection.

See Also

SQLConnect(), SQLSelect()

Writing New Records to a Table

You can insert new records to a database using the `SQLInsert()` function. The `SQLInsert()` function uses the current value of an InTouch tag to insert one record into a table. The `SQLInsert()` function is a one step operation that prepares, inserts, and ends the statement.

If the string associated with 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.

Note InTouch tags cannot be NULL. It is impossible to update or insert NULL values into the database using these functions if the Bind List includes the field. You can insert NULL values into a field using `SQLExecute` on an INSERT statement that does not include the field, which should have been defined to allow NULL values.

SQL Access provides three other functions that separately prepare, insert, and clean up after a record insertion. Using these functions together, you can write scripts that include a single prepare and end statement and add as many record insert statements as needed. If you use individual functions to insert data instead of the `SQLInsert()` function, you can reduce resource usage on the computer.

SQLInsert() Function

The `SQLInsert()` function inserts a new record into the referenced table using the values of the tags in the supplied `BindList`. The `BindList` parameter defines which InTouch tags are used and which database columns they are associated.

Use the `SQLInsert()` function to prepare, insert, and end the statement.

Category

SQL

Syntax

```
[ResultCode=]SQLInsert (ConnectionID, TableName,  
    BindList);
```

Arguments*ConnectionID*

Name of a memory integer tag that holds the number (ID) assigned by the SQLConnect() function to each database connection.

TableName

Name of the database table you want to access.

BindList

Defines which InTouch tags are used and which database columns they are associated with.

Example

The following statement inserts a new record into table ORG with the tag values specified in List1:

```
ResultCode=SQLInsert (ConnectionID, "ORG", "List1");
```

SQLInsertPrepare() Function

The SQLInsertPrepare() function creates and prepares an Insert statement each time the function runs. The Insert statement is not processed. The *StatementID* argument is an integer tag containing a value after the statement is processed.

Category

SQL

Syntax

```
[ResultCode=] SQLInsertPrepare  
    (ConnectionID, TableName, BindList, StatementID);
```

Arguments*ConnectionID*

Name of a memory integer tag that holds the number (ID) assigned by the SQLConnect() function to each database connection.

TableName

The name of the database table to access.

BindList

Defines which InTouch tags are used and which database columns they are associated with.

StatementID

Integer value returned by SQL when a SQLPrepareStatement() function is used.

See Also

SQLConnect(), SQLPrepareStatement()

SQLInsertExecute() Function

The SQLInsertExecute() function runs the previously prepared insert statement specified by the SQLInsertPrepare() function.

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

The *StatementID* argument contains an integer value returned by SQL when a previous SQLInsertPrepare() function is run within the script.

Category

SQL

Syntax

```
[ResultCode=] SQLInsertExecute (ConnectionID,  
    BindList, StatementID);
```

Arguments

ConnectionID

A memory integer tag created by the user to hold the number (ID) assigned by the SQLConnect function to each database connection.

BindList

Defines which InTouch tags are used and which database columns they are associated with.

StatementID

Integer value returned by SQL when a SQLPrepareStatement() function is used.

See Also

SQLConnect(), SQLPrepareStatement()

SQLInsertEnd() Function

The SQLInsertEnd function cleans up resources associated with the *StatementID* function created by SQLInsertPrepare.

The following example shows how multiple insert functions should be specified in a script.

```
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);
```

Category

SQL

Syntax

```
[ResultCode=] SQLInsertEnd ( ConnectionID,  
                          StatementID );
```

Argument

ConnectionID

Name of a memory integer tag that holds the number (ID) assigned by the SQLConnect() function to each database connection.

StatementID

Integer value returned by SQL when a SQLPrepareStatement() function is used.

See Also

SQLConnect(), SQLPrepareStatement()

Updating Existing Records in a Table

SQL Access provides two functions to update table records with values from InTouch tags:

- SQLUpdate()
- SQLUpdateCurrent()

SQLUpdate() Function

The SQLUpdate() function uses the current values of InTouch tags to update all rows in a table that match the condition set by the *WhereExpr* argument.

Category

SQL

Syntax

```
[ResultCode=] SQLUpdate (ConnectionID, TableName,  
    BindList, WhereExpr);
```

Arguments

ConnectionID

Name of a memory integer tag that holds the number (ID) assigned by the SQLConnect() function to each database connection.

TableName

The name of the database table to access.

BindList

Defines which InTouch tags are used and which database columns they are associated with.

WhereExpr

Defines a condition that can be either true or false for any row of the table. The function updates 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 selects all rows whose name column contains the value EmployeeID:

```
name='EmployeeID'
```

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

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

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

```
temperature>350
```

Example

The following statement updates all records in the table BATCH, whose lot number is 65, to the current values of the tags specified in the *BindList* "List1":

```
ResultCode=SQLUpdate (ConnectionID, "BATCH",  
    "List1", "lotno=65");
```

Note Be sure that all records are unique. If identical records exist in a table, all similar records are updated.

See Also

SQLConnect()

SQLUpdateCurrent() Function

The SQLUpdateCurrent() function updates the current row of the logical table using InTouch tags mapped to the table fields by the Bind List specified in SQLSelect() or SQLExecute() function statements. If there are rows that are identical to the current row, all rows are updated.

Up to 54 identical records can be updated at once. If there are too many identical rows to be updated in SQL Access, the SQLUpdateCurrent() function returns an error. The error message is similar to, "Microsoft Cursor Engine: Key column information is insufficient or incorrect. Too many rows were affected by update."

To avoid this error, create a unique key field in the table that makes each row unique. 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 (Access) or an integer field used as the row Identity (SQL Server) be used as the primary key so that SQLUpdateCurrent() function updates only one row at a time. This primary key field does not have to be included in a Bind List.

Category

SQL

Syntax

```
[ResultCode=] SQLUpdateCurrent (ConnectionID) ;
```

Argument

ConnectionID

Name of a memory integer tag that holds the number (ID) assigned by the SQLConnect() function to each database connection.

Example

```
ResultCode=SQLUpdateCurrent (ConnectionID) ;
```

See Also

SQLConnect()

Deleting Records from a Table

You can use two SQL functions to remove records from a database table.

SQL Access provides two functions to delete table records:

- `SQLClearTable()` deletes records from a table.
- `SQLDelete()` deletes records from a table that match a specified condition

SQLClearTable() Function

The `SQLClearTable()` function deletes all records from a table. It does not delete the table from the database.

Category

SQL

Syntax

```
[ResultCode=] SQLClearTable (ConnectionID,  
    "TableName");
```

Arguments

ConnectionID

Name of a memory integer tag that holds the number (ID) assigned by the `SQLConnect()` function to each database connection.

TableName

Name of the table in which all records are cleared.

Example

In the following example, the `SQLClearTable()` function clears all records from the `BATCH1` table.

```
ResultCode=SQLClearTable (ConnectionID, "BATCH1");
```

See Also

`SQLConnect()`, `SQLClearStatement()`

SQLDelete() Function

The SQLDelete() function removes all records from a table that match a condition specified by the *WhereExpr* argument.

Category

SQL

Syntax

```
[ResultCode=]SQLDelete (ConnectionID, TableName,  
    WhereExpr) ;
```

Arguments

ConnectionID

Name of a memory integer tag that holds the number (ID) assigned by the SQLConnect() function to each database connection.

TableName

Name of the table in which records are cleared that meet the condition specified by the *WhereExpr* argument.

WhereExpr

Defines a condition that can be either true or false for any row of the table. The SQLDelete() function deletes only those row records in which the *WhereExpr* condition is true. The expression must be in the following format:

ColumnName *comparison_operator* expression

Note The SQLDelete() function cannot contain a null *WhereExpr* argument.

Example

The following statement deletes all records in the BATCH1 table whose lot number is equal to 65:

```
ResultCode=SQLDelete (ConnectionID, "BATCH1",  
    "lotno=65") ;
```

Note If the column is a character data type, the expression must be in single quotes such as "MachineID='AG_LX7_2'".

See Also

SQLConnect()

Executing Parameterized Statements

Use the `SQLSetStatement()` and the `SQLAppendStatement()` functions to build dynamic queries. The `SQLSetStatement()` function starts a new SQL statement. This can be any valid SQL statement, including the name of a stored procedure. The `SQLAppendStatement()` function continues a SQL statement using the contents of *string*.

SQLSetStatement() Function

The `SQLSetStatement()` function starts a SQL statement buffer using the contents of *SQLStatement*, on the established connection, *ConnectionID*. There can be one SQL Statement buffer per *ConnectionID*. Errors are returned in the function return.

Category

SQL

Syntax

```
[ResultCode=] SQLSetStatement (ConnectionID,
    SQLStatement);
```

Arguments

ConnectionID

Name of a memory integer tag that holds the number (ID) assigned by the `SQLConnect()` function to each database connection.

SQLStatement

Actual SQL statement, see the following examples.

Examples

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

In the following example, the `StatementID` is set to zero so the statement does not have to call `SQLPrepare(Connect_Id, StatementID)` before running the statement. Because the `StatementID` is not created by the `SQLPrepare` to properly end this select, use the `SQLEnd()` function instead of the `SQLClearStatement()` function.

```
SQLSetStatement( Connect_Id, "Select Speed,
    Ser_No from tablename where Ser_No ='" +
    Serial_input + "'");
SQLExecute (Connect_Id, 0);
```

In the following example, the StatementID is created by the SQLPrepareStatement() function and used in the SQLExecute() function. To end this SELECT statement, use the SQLClearStatement() function to free resources and the handle.

```
SQLSetStatement( Connect_Id, "Select Speed,
  Ser_No from tablename where Ser_No =' " +
  Serial_input + "'");
SQLPrepareStatement(Connect_Id,StatementID);
SQLExecute(Connect_Id,StatementID);
SQLSetStatement( Connect_Id, "Select Speed,
  Ser_No from tablename where Ser_No =' " +
  Serial_input + "'");
SQLPrepareStatement(Connect_Id,StatementID);
SQLExecute(Connect_Id,StatementID);
```

See Also

SQLConnect()

SQLAppendStatement() Function

The SQLAppendStatement() function continues a SQL statement using the contents of a string. A return value indicates if an error occurred during the function call.

InTouch tags can support character strings to a maximum of 131 characters. You typically use the SQLAppendStatement() function to concatenate additional strings to a statement.

Category

SQL

Syntax

```
[ResultCode=] SQLAppendStatement( ConnectionID,
  "SQLStatement" );
```

Arguments

ConnectionID

Name of a memory integer tag that holds the number (ID) assigned by the SQLConnect() function to each database connection.

SQLStatement

Actual statement to append.

Example

```
ResultCode=SQLAppendStatement( ConnectionID,
  "where tablename.columnname=TR-773-01" );
```

See Also

SQLConnect(), SQLClearStatement()

Creating a Statement or Loading an Existing Statement from a File

You can create a query with other third-party database tools, and then use SQL Access to run the query. First, you must load the SQL statement from an .SQL query file created by the third-party database tool.

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

You load the SQL query using the `SQLLoadStatement()` function. The statement is now ready to run.

SQLLoadStatement() Function

The `SQLLoadStatement()` function reads a SQL statement from a file.

There can be only one statement per file. However, `SQLAppendStatement()` function can be used to append something to the statement if `SQLPrepareStatement()` function or `SQLExecute()` function has not been called.

Category

SQL

Syntax

```
[ResultCode=] SQLLoadStatement (ConnectionID,  
    FileName);
```

Arguments

ConnectionID

Name of a memory integer tag that holds the number (ID) assigned by the `SQLConnect()` function to each database connection.

FileName

Name of the file containing the SQL statement.

Remarks

After you load the statement and get the statement handle, use the `SQLPrepareStatement()` function to prepare the statement for execution.

Example

The SQL.txt file contains the following SQL statement:

```
Select ColumnName from TableName where  
    ColumnName>100;
```

The SQLLoadStatement() function loads the statement from the file.

```
ResultCode=SQLLoadStatement (ConnectionID,  
    "C:\SQL.txt")
```

See Also

SQLConnect(), SQLAppendStatement(), SQLExecute(), SQLPrepareStatement

Preparing a Statement

Using the following functions, you can create any parameterized statement you want, and then dynamically fill in the parameters one by one. For example, you could save a generic statement in a file, load it using the SQLLoadStatement() function, prepare it using the SQLPrepareStatement() function to get a statement ID, and then fill in the statement parameters using the following functions:

- SQLPrepareStatement()
- SQLSetParamChar()
- SQLSetParamDate()
- SQLSetParamDateTime()
- SQLSetParamDecimal()
- SQLSetParamFloat()
- SQLSetParamInt()
- SQLSetParamLong()
- SQLSetParamNull()
- SQLSetParamTime()
- SQLClearParam()
- SQLClearStatement()

To perform parameter substitution on a SQL statement, place a "?" in the SQL statement where you want to specify a subsequent parameter. The statement is prepared, parameters are set into the statement, and then the statement is run.

SQLPrepareStatement() Function

The SQLPrepareStatement() function prepares the SQL statement to be run. It does not run the statement, it just makes the statement active so you can set parameter values.

Category

SQL

Syntax

```
SQLPrepareStatement(ConnectionId, StatementID)
```

Arguments

ConnectionID

Name of a memory integer tag that holds the number (ID) assigned by the SQLConnect() function to each database connection.

Remarks

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.

Setting Statement Parameters

SQL Access Manager provides a set of functions to modify the value assigned to a parameter included in a SQL statement.

SQLSetParamChar() Function

The SQLSetParmChar() function can be used in a script to set the value of the specified parameter to the specified string. The function can be called multiple times before executing, resulting in the parameter value being set to the concatenation of all values sent. Lengths of 0 (zero) are ignored.

Category

SQL

Syntax

```
SQLSetParamChar(StatementID, ParameterNumber,  
                ParameterValue, MaxLength);
```

Arguments

StatementID

Integer value returned by SQL when a SQLPrepareStatement() function is used.

ParameterNumber

Parameter number in the statement.

ParameterValue

Value to set as the parameter value.

MaxLength

Maximum width 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 *MaxLength* 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.

See Also

SQLPrepareStatement()

SQLSetParamDate() Function

The SQLSetParamDate() function sets the value of a parameter to a specified date.

Category

SQL

Syntax

```
SQLSetParamDate(StatementID, ParameterNumber, "Value");
```

Arguments

StatementID

Integer value that identifies a SQL statement within a query.

ParameterNumber

Integer value that identifies the parameter in the SQL statement identified by the *StatementID* argument.

Value

Date assigned to the parameter as a literal enclosed with double quotation marks or the name of a tag whose value is a date. The time assigned to the date is 12:00:00 am.

Example

This example sets the second parameter of the third statement to the date associated with the NewDate tag.

```
SQLSetParamDate(3, 2, NewDate);
```

See Also

SQLPrepareStatement()

SQLSetParamDateTime() Function

The SQLSetParamDateTime() function sets the value of a parameter to a specified date and time.

Category

SQL

Syntax

```
SQLSetParamDateTime(StatementID, ParameterNumber,  
                    Value, Precision);
```

Arguments

StatementID

Integer value that identifies a SQL statement within a query.

ParameterNumber

Integer value that identifies the parameter in the SQL statement identified by the *StatementID* argument.

Value

Date and time to assign to the parameter identified by the *ParameterNumber* argument.

Precision

Integer that specifies the number of characters of the date-time value assigned as the value of the parameter.

See Also

SQLPrepareStatement()

SQLSetParamDecimal() Function

The SQLSetParamDecimal() function sets the value of a parameter to a decimal number.

Category

SQL

Syntax

```
SQLSetParamDecimal(StatementID, ParameterNumber, Value,  
                  Precision,Scale);
```

Arguments

StatementID

Integer value that identifies a SQL statement within a query.

ParameterNumber

Integer value that identifies the parameter in the SQL statement identified by the *StatementID* argument.

Value

Value can be either a string or an InTouch message tag that represents a decimal number (123.456) 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 DAServer), the function continues to work. But, high precision may not be guaranteed because of the limitation of floating point representation.

Precision

Integer that specifies the total number of digits in the number.

Scale

Integer that specifies the number of digits to the right of the decimal point.

Example

This example sets the second parameter of the third SQL statement to 123.456. The precision is six digits and the scale is three digits to the right of the decimal point.

```
SQLSetParamFloat(3, 2, 123.456, 6, 3)
```

See Also

SQLPrepareStatement()

SQLSetParamFloat() Function

The SQLSetParamFloat() function sets the value of a parameter to a 64-bit, signed, floating-point value.

Category

SQL

Syntax

```
SQLSetParamFloat(StatementID, ParameterNumber, Value);
```

Arguments

StatementID

Integer value that identifies a SQL statement within a query.

ParameterNumber

Integer value that identifies the parameter in the SQL statement identified by the *StatementID* argument.

Value

64-bit, signed, floating-point number to assign as the value of the specified parameter.

Example

This example sets the second parameter of the third SQL statement to -5.

```
SQLSetParamFloat(3, 2, -5)
```

See Also

SQLPrepareStatement()

SQLSetParamInt() Function

The SQLSetParamInt() function sets the value of a parameter to a 16-bit signed integer.

Category

SQL

Syntax

```
SQLSetParamInt(StatementID, ParameterNumber, Value);
```

Arguments

StatementID

Integer value that identifies a SQL statement within a query.

ParameterNumber

Integer value that identifies the parameter in the SQL statement identified by the *StatementID* argument.

Value

16-bit, signed, integer to assign as the value of the specified parameter.

Example

This example sets the second parameter of the third SQL statement to -5.

```
SQLSetParamInt(3, 2, -5)
```

See Also

SQLPrepareStatement()

SQLSetParamLong() Function

The SQLSetParamLong() function sets the value of a parameter to a 32-bit signed analog number.

Category

SQL

Syntax

```
SQLSetParamLong(StatementID, ParameterNumber, Value);
```

Arguments

StatementID

Integer value that identifies a SQL statement within a query.

ParameterNumber

Integer value that identifies the parameter in the SQL statement identified by the *StatementID* argument.

Value

32-bit signed analog number to assign as the value of the specified parameter.

Example

This example sets the third parameter of the first statement to 4.5e12.

```
SQLSetParamLong(1, 3, 4.5e12);
```

See Also

SQLPrepareStatement()

SQLSetParamNull() Function

The SQLSetParamNull() function sets a specified parameter within a SQL statement to NULL.

Category

SQL

Syntax

```
SQLSetParamNull(StatementID, ParameterNumber,  
                ParameterType, Precision, Scale)
```

Arguments

StatementID

Integer value that identifies a SQL statement within a query.

ParameterNumber

Integer value that identifies the parameter in the SQL statement identified by the *StatementID* argument.

ParameterType

Integer value that specifies the type of data associated with the parameter specified by the *ParameterNumber* argument. The *ParameterType* argument can be assigned the following values:

- 0: String
- 1: Date/time
- 2: Integer
- 3: Floating point number
- 4: Decimal number

Precision

Precision of the data associated with the parameter data type.

Scale

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

Remarks

Comparison with the NULL value is controlled by the ANSI_NULLS option in SQL Server. 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() does 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

This transaction set returns all rows of the Products table where the ProductName is not NULL.

```
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 run 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);
```

See Also

SQLPrepareStatement()

SQLSetParamTime() Function

The SQLSetParamTime() function sets the value of the specified time parameter to a specified string.

Category

SQL

Syntax

```
SQLSetParamTime(StatementID, ParameterNumber, Value)
```

Arguments

StatementID

Integer value that identifies a SQL statement within a query.

ParameterNumber

Actual parameter number in the SQL statement identified by the *StatementID* argument.

Value

Actual value to set. Set the parameter specified by the *ParameterNumber* argument to a time value. The current date from the computer running the function is included with the specified time.

Example

This examples sets the second parameter from the fourth SQL statement to 10:00 a.m.

```
ResultCode=SQLSetParamTime( 1, 3, "10:00:00 AM" );
```

See Also

SQLPrepareStatement()

Clearing Statement Parameters

The `SQLClearParam()` function clears the value of the specified parameter.

SQLClearParam() Function

The `SQLClearParam()` function clears the value of the specified parameter. One of the `SQLSetParamxxx()` functions must be called again to reload parameters before calling the `SQLExecute()` function to run the query.

Category

SQL

Syntax

```
[ResultCode=]SQLClearParam(StatementID,ParameterNumber);
```

Arguments

StatementID

Integer value returned when a `SQLPrepareStatement()` function runs.

ParameterNumber

The *ParameterNumber* argument identifies the actual argument within the SQL statement to modify. Set the value of *ParameterNumber* associated with *StatementID* to zero or a zero-length string, depending on whether the argument is numeric or a string.

See Also

`SQLPrepareStatement()`, `SQLExecute()`

Executing a Statement

The `SQLExecute()` function can be used within an InTouch script to run a SQL query during run time.

SQLExecute() Function

The `SQLExecute` function runs a SQL query within a script. If the statement includes a `SELECT`, the *BindList* argument designates the name of the Bind List to use for binding the database columns with InTouch tags. If the Bind List is `NULL`, no tag associations are made.

Category

SQL

Syntax

```
SQLExecute (ConnectionID, BindList, StatementID);
```

Arguments

ConnectionID

Name of a memory integer tag that holds the number (ID) assigned by the `SQLConnect()` function to each database connection.

BindList

The *BindList* argument 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* argument. A zero-length Bind List is useful when it is known in advance that the *StatementID* is not associated with a row-returning query.

StatementID

Integer value returned by SQL when a `SQLPrepareStatement()` function is used.

Remarks

Errors are returned in the function return. If the statement has been prepared, the statement handle returned from the prepare should be passed. If the statement has not been prepared, the statement handle should be zero.

Note The `SQLExecute()` function can be called only once for a statement that has not been prepared. If the statement has been prepared, it can be called multiple times.

A default statement is 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 the `SQLLoadStatement()`, `SQLSetStatement()`, and `SQLAppendStatement()` functions. The default statement is used by `SQLExecute()` whenever `StatementID = 0` is specified.

Examples

This example loads the SQL statements from the `lotquery.sql` file and places the results of the `SELECT` statement to InTouch tags specified by the Bind List.

```
ResultCode = SQLLoadStatement (ConnectionID,
    "c:\myappdir\lotquery.sql");
ResultCode = SQLExecute (ConnectionID, "BindList", 0);
ResultCode = SQLNext (ConnectionID);
```

This `SQLSetStatement()` function must be used for complex queries and string expressions greater than 131 characters. When the string expression exceeds 131 characters use the `SQLAppend()` function.

```
SQLSetStatement(ConnectionID, "Select Speed, Ser_No
    from tablename where Ser_No =" + Serial_input +
    "");
SQLExecute(ConnectionID, "BindList", 0);
```

In the previous example, the `StatementID` argument is set to zero so the statement does not have to call `SQLPrepareStatement(Connection_Id, StatementID)` before the execute statement.

Because the `StatementID` is not created by the `SQLPrepare` statement to properly end this `SELECT`, use the `SQLEnd()` function instead of the `SQLClearStatement()` function.

```
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 a `SQLPrepareStatement` function call and used by the `SQLExecute` function. To end this `SELECT` statement, use a `SQLClearStatement()` function call within a script to free resources and the `StatementID`.

The `SQLExecute()` function supports 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 write the InTouch QuickScript to run the stored procedure based upon the type of database that you are using. The following example shows script statements to run a stored procedure for a SQL Server database.

```
ResultCode = SQLSetStatement
    (ConnectionID, "LotInfoProc");
ResultCode = SQLExecute(ConnectionID, "BindList", 0);
ResultCode = SQLNext (ConnectionID);
{Get results of Select}
```

The following example shows script statements to run a stored procedure for an Oracle database.

```
ResultCode = SQLSetStatement (ConnectionID, "{CALL
    LotInfoProc}");
ResultCode = SQLExecute(ConnectionID, "BindList", 0);
ResultCode = SQLNext (ConnectionID);
{Get results of Select}
```

See Also

SQLConnect(), SQLPrepareStatement()

Releasing Occupied Resources

The SQLClearStatement function releases database resources associated with the statement specified by the *StatementID*.

SQLClearStatement() Function

The SQLClearStatement() function releases database resources associated with the statement specified by the *StatementID* argument.

Category

SQL

Syntax

```
[ResultCode=]SQLClearStatement (ConnectionID,
    StatementID);
```

Arguments

ConnectionID

Name of a memory integer tag that holds the number (ID) assigned by the SQLConnect() function to each database connection.

StatementID

Integer value returned by SQL when a SQLPrepareStatement() function is used.

See Also

SQLConnect(), SQLPrepareStatement()

Working with Transaction Sets

SQL Access includes a set of transaction functions to change insert, update, or delete records from a database. Generally, these transactions are grouped within a script in the form of a transaction set. A transaction set is committed at one time.

SQLTransact() Function

The SQLTransact() function defines the beginning of a group of SQL statements called a transaction set. A transaction set is handled like a single transaction. After the SQLTransact() function runs, all subsequent operations are not committed to the database until the SQLCommit() function runs successfully.

Category

SQL

Syntax

```
[ResultCode=]SQLTransact (ConnectionID)
```

Argument

ConnectionID

Name of a memory integer tag that holds the number (ID) assigned by the SQLConnect() function to each database connection.

Example

This example transaction set includes three insert statements.

```
ResultCode = SQLTransact( ConnectionID );  
ResultCode = SQLInsertPrepare( ConnectionID, TableName,  
    BindList, StatementID );  
ResultCode = SQLInsertExecute( ConnectionID, BindList,  
    StatementID );  
ResultCode = SQLInsertExecute( ConnectionID, BindList,  
    StatementID );  
ResultCode = SQLInsertExecute( ConnectionID, BindList,  
    StatementID );  
ResultCode = SQLInsertEnd( ConnectionID, StatementID );  
ResultCode = SQLCommit( ConnectionID );
```

See Also

SQLCommit(), SQLRollback()

SQLCommit() Function

The SQLCommit() function defines the end of a transaction set. After the SQLTransact() function runs, all subsequent all SQL statements within the transaction set are not committed to the database until the SQLCommit() function runs successfully.

Note Use caution when writing QuickScripts that include the SQLCommit() function. Processing time increases with the number of SQL statements in a transaction set.

Category

SQL

Syntax

```
[ResultCode=]SQLCommit (ConnectionID)
```

Argument

ConnectionID

Name of a memory integer tag that holds the number (ID) assigned by the SQLConnect() function to each database connection.

Example

This example script includes a transaction set that makes three database inserts.

```
ResultCode = SQLTransact( ConnectionID);  
ResultCode = SQLInsertPrepare( ConnectionID, TableName,  
    BindList, StatementID );  
ResultCode = SQLInsertExecute( ConnectionID, BindList,  
    StatementID );  
ResultCode = SQLInsertExecute( ConnectionID, BindList,  
    StatementID );  
ResultCode = SQLInsertExecute( ConnectionID, BindList,  
    StatementID );  
ResultCode = SQLInsertEnd( ConnectionID, StatementID );  
ResultCode = SQLCommit( ConnectionID);
```

See Also

SQLRollback(), SQLTransact(), SQLCommit()

SQLRollback() Function

The SQLRollback() function reverses or rolls back the most recent transaction set. A transaction set is a group of commands issued between the SQLTransact() and the SQLCommit() functions.

A transaction set is handled like a single transaction. After the `SQLTransact()` function runs, all subsequent operations are not committed to the database. Query changes to the database occur after the `SQLCommit()` function runs. The `SQLRollback()` function rolls back the transaction set if it runs before the `SQLCommit()` function.

Category

SQL

Syntax

```
[ResultCode=]SQLRollback(ConnectionID)
```

Argument*ConnectionID*

Name of a memory integer tag that holds the number (ID) assigned by the `SQLConnect()` function to each database connection.

Example

This example rolls back the database values prior to the `SQLTransact` function within the script.

```
ResultCode =SQLTransact( ConnectionID);  
ResultCode = SQLInsertPrepare( ConnectionID, TableName,  
    BindList, StatementID );  
ResultCode = SQLInsertExecute( ConnectionID, BindList,  
    StatementID );  
ResultCode = SQLInsertEnd( ConnectionID, StatementID );  
ResultCode =SQLRollback( ConnectionID);
```

See Also`SQLCommit()`, `SQLTransact()`

Opening the ODBC Administrator Dialog Box at Run Time

Use the `SQLManageDSN()` function to run the Microsoft ODBC Manager while an InTouch application is running.

SQLManageDSN() Function

The `SQLManageDSN` function runs the Microsoft ODBC Manager setup program while an InTouch application is running. `SQLManageDSN()` can be used within a script to add, delete, and modify existing data source names of a SQL Server or Access database.

Category

SQL

Syntax

```
SQLManageDSN(ConnectionId)
```

Argument

ConnectionId

ConnectionId is not used. It is retained for backward compatibility with older versions of SQL Access. Therefore, any number can be passed to the function. No database connection needs to be established before running the function to open Microsoft ODBC Manager.

Example

```
SQLManageDSN( 0 );
```

Understanding SQL Error Messages

This section explains how to troubleshoot InTouch applications that use SQL Access 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.

SQLErrorMsg() Function

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 and assigns it as the value of an InTouch message tag.

Category

SQL

Syntax

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

Argument

Result Code

Integer value returned by a previous SQL function. The `SQLErrorMsg()` function sets the value of an InTouch message tag to the message associated with the result code. For more information about error messages associated with result codes, see [Understanding SQL Error Messages](#) on page 181.

Remarks

Refer to your database documentation for undocumented result codes. Also, browse the Wonderware Log Viewer for any additional error messages.

The `SQLTrace=1` flag defined under the `[InTouch]` section of the `win.ini` file is useful for debugging SQL Access scripts.

Example

This example assigns the error message associated with the SQL Access Manager result code to the `ErrorMsg` message tag.

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

See Also

`SQLConnect()`

SQL Access Manager Result Codes and Messages

The following table lists some common SQL Access result codes, their corresponding error messages, and descriptions:

Result Code	Error Message	Description
0	No errors occurred	The SQL function ran successfully without errors.
-1	<Message from the database provider>	A specific error message from the vendor database.
-2	An empty statement cannot be executed	A <code>SQLExecute(ConnectionId, BindList, 0)</code> is run without previously calling <code>SQLSetStatement</code> or <code>SQLLoadStatement</code> 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 message sent to the Wonderware Log Viewer.
-5	No more rows to fetch	The last record in the table has been reached.
-7	Invalid parameter ID	The <code>SQLSetParamChar()</code> , <code>SQLSetParamDate()</code> , <code>SQLSetParamDateTime()</code> , <code>SQLSetParamDecimal()</code> , <code>SQLSetParamFloat()</code> , <code>SQLSetParamInt()</code> , <code>SQLSetParamLong()</code> , <code>SQLSetParamNull()</code> , or <code>SQLSetParamTime()</code> function 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	The <code>SQLSetParamNull</code> function is called with an invalid <i>Type</i> argument value.
-10	Changing data type of parameter is not allowed	The <code>SQLSetParamChar()</code> , <code>SQLSetParamDate()</code> , <code>SQLSetParamDateTime()</code> , <code>SQLSetParamDecimal()</code> , <code>SQLSetParamFloat()</code> , <code>SQLSetParamInt()</code> , <code>SQLSetParamLong()</code> , <code>SQLSetParamNull()</code> , or <code>SQLSetParamTime()</code> function is called with a different type for an existing parameter.

Result Code	Error Message	Description
-11	Adding parameter after the statement has been executed successfully is not allowed.	The SQLSetParamChar(), SQLSetParamDate(), SQLSetParamDateTime(), SQLSetParamDecimal(), SQLSetParamFloat(), SQLSetParamInt(), SQLSetParamLong(), SQLSetParamNull(), or SQLSetParamTime() function is called for a new parameter ID after the statement has been run 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 run this function.
-1002	Invalid connection	The value passed to the <i>ConnectionId</i> argument 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.
-1006	String is null	Warning - the string read from the database is null. This is only a warning message sent to the Wonderware Log Viewer.
-1007	String is truncated	Warning - the string read from the database is longer than 131 characters and is truncated when selected. The warning is sent to the Wonderware Log Viewer.

Result Code	Error Message	Description
-1008	No Where clause	There is no Where clause on Delete.
-1009	Connection failed	Check the Wonderware Log Viewer for more information about the failed connection to the database.
-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() function is called without running a SQLSelect() or SQLExecute() function first.
-1013	Unable to find file to load	The SQLLoadStatement() function is called with a file name that cannot be found.

Error messages from a vendor database return a *ResultCode* of -1. The SQL Access function *ResultCode* is always -1, but the message is copied exactly from the database provider.

For error messages that occur when using an Oracle database, refer to Oracle Server documentation for specific error messages and solutions.

The following table lists common error messages that can occur when using a Microsoft SQL Server or Access database.

Error Message	Solution
You cannot have more than one statement active at a time	You are trying to run a SQL command after calling the SQLSelect() function. Run SQLEnd() to free system resources from the SQLSelect() or use a separate <i>ConnectionId</i> 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 your Microsoft SQL Server documentation for specific error messages and solutions.

Reserved Word List

This section lists keywords that are excluded from use with the SQL Access Bind List, the Table Template, and the ODBC interface.

If a reserved keyword is used as the Column Name in a Bind List or Table Template, an error message appears in the Wonderware Log Viewer. The type of error depends upon the ODBC driver being used and the location where the keyword is found. For example, one of the most common errors 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."

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. If the SQL command cannot be interpreted correctly, SQL Access Manager generates an error message that can be viewed from the Wonderware Log Viewer.

The following alphabetical list shows the reserved keywords for SQL Access and ODBC:

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

DICTIONARY	DISCONNECT	DISPLACEMENT
DISTINCT	DOMAIN	DOUBLE
DROP	ELSE	END
ESCAPE	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	INSERT
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
VALUE	VALUES	VARCHAR
VARING	VIEW	WHEN
WHENEVER	WHERE	WITH
WORK	YEAR	

Chapter 4

Using the 16-Pen Trend Wizard

You can use an InTouch wizard to create real-time and historical trends capable of displaying data from up to 16 tags. The 16-Pen Trend is a supplementary component that you can install during an InTouch installation.


The 16-Pen Trend Wizard can be configured much like other InTouch chart wizards. The 16-Pen Trend wizard allows you to configure the following trend properties:

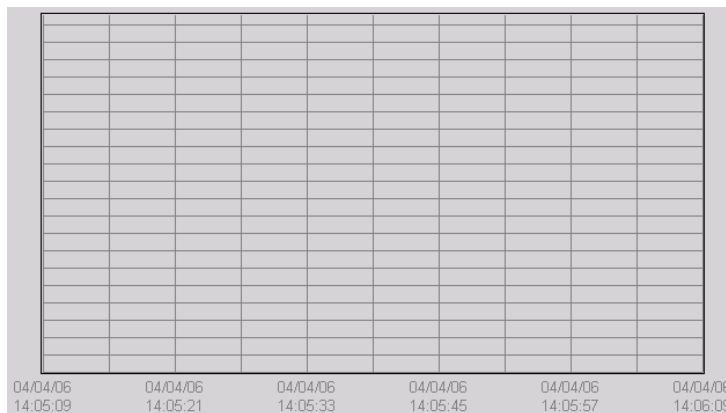
- Tag assigned to each trend pen
- Trend line width and color
- Starting and ending dates and times for historical trends
- Update rate and time span for real-time trends
- Minimum and maximum engineering units assigned to a trend tag
- Major and minor trend time divisions
- Major and minor trend value divisions

Creating a 16-Pen Trend

You can create a trend by selecting the 16-Pen Trend Wizard from WindowMaker.

To create a 16-Pen real-time or historical trend

- 1 Open a window from WindowMaker to place the 16-Pen Trend.
- 2 Click the wizard tool in the Wizard Toolbar. The **Wizard Selection** dialog box appears.
- 3 Select **Trends** from the list of wizards. The right pane of the **Wizard Selection** dialog box shows a set of trend wizard icons.
- 4  Select the 16-Pen Trend wizard and click **OK**. The **Wizard Selection** dialog box closes and your window reappears.
- 5 Click in the window to place the 16-Pen Trend. The wizard places a 16-Pen Trend template in the window.



- 6 Double-click the 16-Pen Trend template to open the **PenTrend Control** dialog box.
- 7 In the **Trend Type** area, select **Historical** or **Realtime** as the type of trend you want to create.

Trend Type	Options
<input type="radio"/> Historical <input checked="" type="radio"/> Realtime	<input checked="" type="checkbox"/> Enable runtime configuration

The **PenTrend Control** dialog box automatically shows the appropriate time and update options based upon the type of trend you select.

- 8 In the **Options** area, select or clear the **Enable runtime configuration** option.

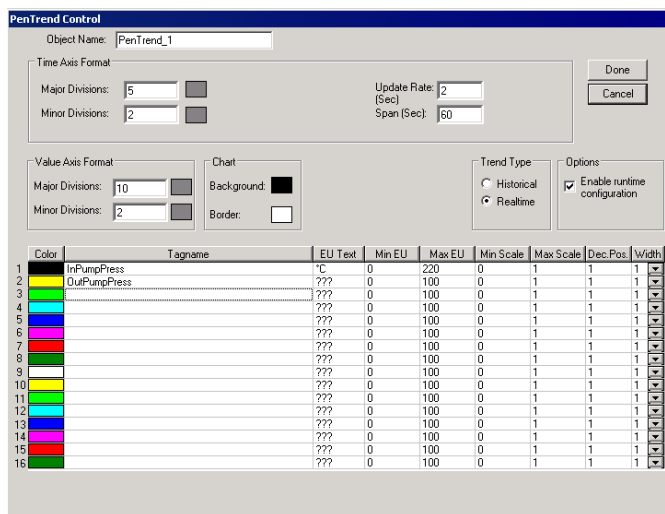
Selecting this option allows operators to modify some properties of the 16-Pen Trend while it is running.

Configuring Which Tags to Display on the Trend Graph

You can use the 16-Pen Trend Wizard to assign tags to trend pens. The 16-Pen Trend Wizard includes a set of columns that specify tag properties shown on the trend. These columns use the default property values assigned to the tag from the Tagname Dictionary. You can override these assigned tag values by specifying other values when you configure the trend.

To configure 16-Pen Trend tags

- 1 If needed, open the window containing the 16-Pen Trend template.
- 2 Double-click the 16-Pen Trend. The **PenTrend Control** dialog box appears with a grid area near the bottom to specify the tags associated with trend pens.



- 3 In the **Object Name** box, assign a name to the 16-Pen Trend.
The default name is PenTrend_1, which increments the number in the name as you create each new trend.
- 4 In the **Tagname** box, enter the name of the tag to associate with the pen number listed at the left of the grid.
Double-clicking within a cell beneath **Tagname** shows the **Select Tag** dialog box. You can assign a tag to a pen by selecting the tag from the **Select Tag** dialog box.

Note You remove a tag by selecting a **Tagname** box containing a tag name and pressing your keyboard space bar.

- 5 In the **Color** column, click each color box to open a color palette. Select a color for the pen.

- 6** In the **EU Text** column, enter the text that you want to initially use in run time as the header text for the pen axis for each respective pen.

This text is the axis text when a pen is set to active. The **EU Text** column is initially assigned the tag's engineering units from the Tagname Dictionary. You can override these default engineering units for the 16-Pen Trend.
- 7** In the **Min EU** column, enter the minimum engineering units value assigned to the pen.

The **Min EU** column initially shows the tag's minimum engineering units value from the Tagname Dictionary. You can assign another minimum engineering units value that applies only to a 16-Pen Trend.
- 8** In the **Max EU** column, enter the maximum engineering units assigned to the pen.

The **Max EU** column initially shows the tag's maximum engineering units value from the Tagname Dictionary. You can assign another maximum engineering units value that applies only to a 16-Pen Trend.

Note The Min/Max engineering units are very important for showing historical trend data. The historical trend shows from 0-100 percent of engineering units scale.

- 9** In the **Min Scale** column, enter the percentage that you want to use initially in run time to calculate the minimum pen axis grid for the respective EU scale.
- 10** In the **Max Scale** column, enter the percentage that you want to use initially in run time to calculate the maximum pen axis grid for the respective EU scale.
- 11** In the **Dec.Pos** column, enter the number of decimal points that you want to use initially in run time when labeling the pen axis grid.
- 12** In the **Width** column, select the pen line width in pixels to plot data values shown on the trend.
- 13** Continue with the next procedure to update the trend time and update rate of a 16-Pen Trend.

Configuring the Trend Time Span and Update Rate

The **Pen Trend Control** dialog box shows different options based upon whether you are creating a real-time or historical 16-Pen Trend. You can set the time span for a historical trend and the update rate for a real-time trend.

To configure the time span of a 16-Pen historical trend

- 1 Double-click a 16-Pen historical trend within a window. The **Pen Trend Control** dialog box appears with options to set the starting and ending dates and time of a trend.



Start Time: 4/4/2006 2:07:47 PM
End Time: 4/4/2006 2:08:47 PM

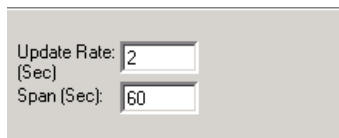
- 2 Set the starting and ending date and time of the historical trend.

Use the following format for both the starting and ending dates and times:

MM/DD/YY HH:MM:SS AM/PM

To configure the update rate of a 16-Pen real-time trend

- 1 Double-click a 16-Pen real-time trend object within a window. The **Pen Trend Control** dialog box appears with options to set update rate and span of a real-time trend.



Update Rate (Sec): 2
Span (Sec): 60

- 2 In the **Update Rate** box, type the number of seconds between each refresh interval of the historical trend.
- 3 In the **Span** box, type the number of seconds of the real-time interval shown in the trend.

Configuring the Trend Display Options

You can configure the visual appearance of a trend with the 16-Pen Trend Wizard.

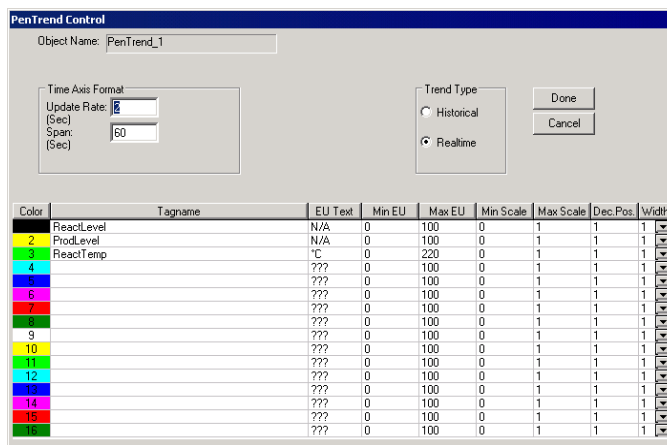
To configure the display options of a 16-Pen Trend

- 1 Double-click the 16-Pen Trend in WindowMaker. The **Pen Trend Control** dialog box appears.
- 2 In the **Time Axis Format** area, enter the number of major time divisions in **Major Divisions**. This option sets the number of major time divisions on the horizontal axis of the trend.
- 3 Click the color box to the right of **Major Divisions** to open the color palette and select a color if you want to assign another color to the major time division lines. Otherwise, skip this step and accept the default color assigned to major time division lines.
- 4 In the **Minor Divisions** box, enter the number of minor time divisions shown on the horizontal axis of a trend.
- 5 Select a color for the minor time division lines.
- 6 In the **Value Axis Format** area, enter the number of major divisions in **Major Divisions**. This option sets the number of major divisions shown on the vertical value axis.
- 7 Set the color of the major value divisions.
- 8 In the **Minor Divisions** box, enter the number of minor value divisions shown on the vertical axis of the trend.
- 9 Set the color of the minor value divisions.
- 10 In the **Chart** area, select the background and border colors of the trend.
- 11 Click **Done** to save the configuration changes to the 16-Pen Trend.

Changing the Trend Configuration at Run Time

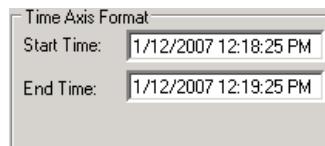
If the **Enable runtime configuration** option is selected from the **PenTrend Control** dialog box, operators can change some characteristics of a 16-Pen Trend while the application is running.

Run-time changes to the 16-Pen Trend are not permanent. If operators close WindowViewer and then start the application window again, the 16-Pen Trend retains the configuration originally defined from WindowMaker. The following figure shows the **PenTrend Control** dialog box that appears if you click on a 16-Pen Trend while it is running.

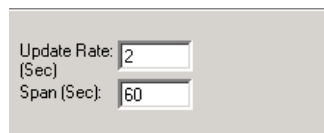


You can change the following during run time:

- Tags or expressions assigned to trend pens
- Characteristics of trend tags or expressions
- Type of trend (Historical or Realtime)
- Date and time range of a historical trend



- Update rate and frequency of a real-time trend



After clicking **Done**, the trend retains the configuration changes for the duration of the current WindowViewer application session.

Controlling a 16-Pen Trend Wizard Using Scripts

You can use a set of functions in QuickScripts to control a 16-Pen Trend object in run time. For example, you can connect pens to the chart, add new events to the chart, remove or replot the grid, and remove or replot the scooters.

ptGetTrendType() Function

The `ptGetTrendType()` function can be used in a script to return a value that indicates whether the current mode of a 16-Pen Trend shows historical or real-time data.

Category

Pen trend

Syntax

```
ptGetTrendType (TrendName) ;
```

Argument

TrendName

Name of the trend. *TrendName* must be either a string constant or message tag.

Return Value

Returns the trend type:

- 0 = Historical trend
- 1 = Real-time noscroll
- 2 = Real-time trend

Example

The following example returns a value that indicates whether the PumpPress trend shows historical or real-time data.

```
ptGetTrendType ("PumpPress") ;
```

ptLoadTrendCfg() Function

The `ptLoadTrendCfg()` function can be used in a script to load trend configuration values from a file.

Category

Pen trend

Syntax

```
ptGetTrendCfg (TrendName, FileName);
```

Arguments

TrendName

The name of the trend object. The value assigned to *TrendName* must be either a string constant or a message tag.

FileName

Name of the configuration file. The folder path to the configuration file must be included with the *Filename* argument.

Example

The TankFarm trend is configured with values from the `c:\TrendCfg.txt` file.

```
ptLoadTrendCfg ("TankFarm", "C:\TrendCfg.txt");
```

ptPanCurrentPen() Function

The `ptPanCurrentPen()` function can be used in a script to scroll a 16-Pen Trend's pen upward or downward on the vertical value axis. Vertical scrolling is determined by the number of major and minor trend units specified as argument values.

Category

Pen trend

Syntax

```
ptPanCurrentPen(TrendName, MajorUnits, MinorUnits);
```

Arguments

TrendName

The name of the trend object. *TrendName* must be either a string constant or message tag.

MajorUnits

Multiplier to scroll by the number of units defined by the major division lines. A negative number indicates a downward scroll of the vertical axis.

MinorUnits

Multiplier for additional scrolling by number of units defined by the minor division lines. A negative number indicates a downward scroll of the vertical axis.

Examples

This example scrolls the pen upward one major division line.

```
ptPanCurrentPen("TrendName", 1, 0);
```

This example scrolls the pen upward half a minor trend division.

```
ptPanCurrentPen("TrendName", 0, 0.5);
```

This example scrolls the pen downward by 2 major division lines and half of a minor division line.

```
ptPanCurrentPen("TrendName", -2, -0.5);
```

This example scrolls one major division line up 1 and downward by 2 minor division lines.

```
ptPanCurrentPen("TrendName", 1, -2);
```

ptPanTime() Function

The `ptPanTime()` function can be used in a script to scroll a 16-Pen Trend's pen left or right on the horizontal time axis based on the number of specified major or minor trend units.

Category

Pen trend

Syntax

```
ptPanTime (TrendName, MajorUnits, MinorUnits);
```

Arguments

TrendName

The name of the trend object. *TrendName* must be either a string constant or message tag.

MajorUnits

Multiplier for scrolling by the number of horizontal major division lines. A negative number indicates panning left on the trend.

MinorUnits

Multiplier for additional scrolling by number of units defined by the minor division lines. A negative number indicates panning left on the trend.

Remarks

The settings for Major Division and Minor Division specified in the **PenTrend Control** dialog box during development are the basis from which the amount to scroll by is calculated. A trend with a time span of 120 seconds, a major division value of 10 and a minor division value of 2, results in a trend with a major division line every 12 seconds and a minor division line every 6 seconds. The function

```
ptPanTime("TrendName",1,0.5)
```

scrolls the time axis by $1 * 12 + 0.5 * 6 = 15$ seconds.

Examples

This example scrolls the pen 1 major division to the right on the horizontal trend axis.

```
ptPanTime ("TrendName", 1, 0);
```

This example scrolls the pen to the right on the horizontal axis of the trend by 0.5 minor division.

```
ptPanTime ("TrendName", 0, 0.5);
```

This example scrolls the pen 2.5 major divisions to the left on the horizontal axis of the trend.

```
ptPanTime("TrendName", -2, -0.5);
```

This example scrolls the pen 1 major division to the right and 2 minor divisions to the left.

```
ptPanTime("TrendName", 1, -2);
```

ptPauseTrend() Function

The `ptPauseTrend()` function can be used in a script to temporarily stop a 16-Pen Trend from updating the graph. The trend remains stopped until you call `ptPauseTrend` again with a value of 0.

Category

Pen trend

Syntax

```
ptPauseTrend(TrendName, Value);
```

Arguments

TrendName

The name of the trend object. *TrendName* must be either a string constant or message tag.

Value

A value of 1 pauses trend updates. A value of 0 resumes trend updating.

Example

This example pauses any further updates to the 16-Pen Trend while the *Value* argument is 1.

```
ptPauseTrend ("TrendName",1);
```


ptSaveTrendCfg() Function

The `ptSaveTrendCfg()` function can be used in a script to save a trend's current configuration values to a file.

Category

Pen trend

Syntax

```
ptSaveTrendCfg (TrendName, FileName) ;
```

Arguments

TrendName

The name of the trend object. Must be either a string constant or message tag.

FileName

Name of the file to save the trend's configuration values. The folder path to the configuration file can be specified with the `Filename` argument.

Example

The `ptSaveTrendCfg()` function saves the values from the `PumpTrend` 16-Pen Trend to the `c:\Config.txt` file.

```
ptSaveTrendCfg ("PumpTrend", "C:\Config.txt")
```

ptSetCurrentPen() Function

The `ptSetCurrentPen()` function can be used in a script to select a pen by its assigned number to control the pen axis.

Category

Pen trend

Syntax

```
ptSetCurrentPen (TrendName, PenNum) ;
```

Arguments

TrendName

Name of the trend. Must be either a string constant or message tag.

PenNum

Number of the pen (1-16) assigned as the current trend pen.

Example

The `ptSetCurrentPen()` function assigns pen 2 as the current pen of the `PumpPress` trend.

```
ptSetCurrentPen ("PumpPress", 2) ;
```

ptSetPen() Function

The ptSetPen() function can be used in a script to assign a tag to a trend pen.

Category

Pen trend

Syntax

```
ptSetPen (TrendName, PenNum, TagName) ;
```

Arguments

TrendName

The name of the trend object. Must be a string constant or a message tag.

PenNum

Number of the pen assigned as the current trend pen.

TagName

Name of the tag assigned to the trend pen.

Example

The ptSetPen() function assigns the PumpInP tag to pen 2 of the PumpPress trend.

```
ptSetPen ("PumpPress", 2, "PumpInP") ;
```

ptSetPenEx() Function

The `ptSetPenEx()` function can be used in a script to assign a tag to a specific trend pen and override the tag's configuration values specified in the Tagname Dictionary.

Category

Pen trend

Syntax

```
ptSetPenEx (TrendName, PenNum, TagName, minEu, maxEU,
            minPercent, maxPercent, Decimal, EU);
```

Arguments

TrendName

The name of the trend object. Must be either a string constant or message tag.

PenNum

Number of the pen assigned as the current trend pen.

TagName

Name of the tag assigned to the trend pen.

minEU

The minimum engineering units value for the specified tag.

maxEU

The maximum engineering units value for the specified tag.

minPercent

The percentage to use initially in run time to calculate the minimum pen axis grid for the respective EU scale.

maxPercent

The percentage to use initially in run time to calculate the maximum pen axis grid for the respective EU scale.

Decimal

Decimal precision of a tag's value in the trend.

EU

The label for the tag's engineering units.

Example

The `ptSetPenEx()` function assigns the `PumpInP` tag to pen 2 of the `PumpPress` trend. The tag's engineering units range is set between 0 to 1500 and its units are PSI. The percentages for the grid are 0 to 1, and the decimal precision is set to 2.

```
ptSetPenEx ("PumpPress", 2, "PumpInP", 0, 1500, 0, 1, 2,
            "PSI");
```

ptSetTimeAxis() Function

The `ptSetTimeAxis()` function can be used in a script to set the trend's starting date and time and ending date and time.

Category

Pen trend

Syntax

```
ptSetTimeAxis (TrendName, StartDateTime, EndDateTime) ;
```

Arguments

TrendName

The name of the trend object. Must be either a string constant or message tag.

StartDateTime

The date and time when the trend begins. The format for the starting date and time is: mm/dd/yyyy hh:mm:ss AM/PM

EndDateTime

The date and time when the trend ends. The format for the ending date and time is: mm/dd/yyyy hh:mm:ss AM/PM.

Example

The `ptSetTimeAxis()` function sets the starting and ending dates and times of a trend for a 25 hour period starting at 8:30 on May 22, 2007.

```
ptSetTimeAxis ("PumpPress", "05/22/2007 08:30:00 AM",  
              "05/23/2007 09:30:00 AM");
```

ptSetTimeAxisToCurrent() Function

The `ptSetTimeAxisToCurrent()` function can be used in a script to calculate the current chart span and the chart's ending time.

Category

Pen trend

Syntax

```
ptSetTimeAxisToCurrent (TrendName) ;
```

Argument

TrendName

The name of the trend object. *TrendName* must be a string constant or a message tag.

Example

The `ptSetTimeAxisToCurrent()` function sets the ending date and time of the PumpPress trend to the current date and time.

```
ptSetTimeAxisToCurrent ("PumpPress") ;
```

ptSetTrend() Function

The `ptSetTrend()` function can be used in a script to pause or restart updates to a 16-Pen Trend.

Category

Pen trend

Syntax

```
ptSetTrend (TrendName, EnableUpdates) ;
```

Arguments

TrendName

The name of the trend object. Must be either a string constant or message tag.

EnableUpdates

The value 1 starts updates to the trend. The value 0 stops trend updates.

Example

The `ptSetTrend()` function updates the PumpPress trend.

```
ptSetTrend ("PumpPress", 1) ;
```

ptSetTrendType() Function

The `ptSetTrendType()` function can be used in a script to specify whether the trend shows historical or real-time data.

Category

Pen trend

Syntax

```
ptSetTrendType (TrendName, TrendType);
```

Arguments

TrendName

The name of the trend object. Must be either a string constant or Message tag.

TrendType

The value 1 indicates a historical trend. The value 2 specifies a real-time trend.

Example

The `ptSetTrendtype()` function specifies the PumpPress trend shows real-time data.

```
ptSetTrendType ("PumpPress", 2);
```

ptZoomCurrentPen() Function

The `ptZoomCurrentPen()` function can be used in a script to change the value range shown on a trend's Y-axis. The range of the trend's vertical value axis can be increased or decreased by a specified zoom ratio.

Category

Pen trend

Syntax

```
ptZoomCurrentPen (TrendName, ZoomFactor);
```

Arguments

TrendName

The name of the trend object. Must be either a string constant or message tag.

ZoomFactor

Assigning a number larger than 1.0 increases the value range of the trend by multiplying the current range limits by the zoom factor. Assigning a zoom factor less than 1.0 decreases the value range shown in the vertical axis of the trend.

Remarks

The zoom ratio is applied to the existing span of the current pen's Y-axis range. For example, if the trend starts with a Y-axis range of -50 to 50 and then you zoom by a ratio of 2.0, the new range is -100 to 100. If you zoom by 2.0 again, then the new range is -200 to 200. The zoom ratio applies to the range currently in effect, not the original Y-axis range.

The zoom ratio persists during run time for each of the trend's pens. As you switch from one pen to another using the `ptSetCurrentPen()` function, the Y-axis value range reflects the current scaling for the selected pen.

Example

The `ptZoomCurrentPen` function doubles the Y-axis range of the current tag in the trend named "PumpPress".

```
ptZoomCurrentPen("PumpPress", 2);
```

ptZoomTime() Function

The `ptZoomTime()` function can be used in a script to change the time range shown on the trend's horizontal axis.

Category

Pen trend

Syntax

```
ptZoomTime(TrendName, Zoom);
```

Arguments*TrendName*

The name of the trend object. Must be either a string constant or message tag.

Zoom

Assigning a number larger than 1.0 increases the time period shown on the trend's horizontal axis. Assigning a number less than 1.0 decreases the time period shown on the horizontal axis.

Examples

The `ptZoomTime()` function increases the time period shown on the trend's horizontal axis by 17 percent.

```
ptZoomTime("PenTrend_1", 1.17);
```

The `ptZoomTime()` function decreases the time period shown on the trend's horizontal axis by 50 percent. For example, the `ptZoomTime()` function reduces the trend's time period to 30 minutes if the original time range was set to 1 hour.

```
ptZoomTime("PenTrend_1", 0.5);
```

Chapter 5

Symbol Factory

Symbol Factory includes a collection of over 4,000 industrial symbols that can be used as visual elements in your InTouch application windows. Symbol Factory is a supplementary component that you can install during the InTouch HMI installation.

Note Use the ArchestrA Symbol Editor to create visual elements for InTouch applications that interact with Application Server. You can also use the Symbol Editor to create intelligent visual elements for applications independent of the InTouch HMI. For more information about the ArchestrA Symbol Editor, see the Application Server documentation.

Wonderware provides no warranty of any kind for any of this product. You can report problems to Wonderware Technical Support. We highly recommend that you always back up your application and data before you install or use any new utility or application.

Symbol Types

Symbol Factory includes four types of wizards:

- Picture Wizards
- Bitmap Wizards
- Texture Wizards
- InTouch Objects

Picture Wizards

Symbol Factory picture wizards are vector-based images of equipment or flow diagrams. As you create your application, you can modify picture wizard images by doing the following:

- Assign an animation to an image
- Flip an image horizontally or vertically
- Change the horizontal and vertical perspective of an image
- Rotate an image on its axis
- Change the fill color and pattern of an image
- Change the size, pattern, and color of image lines

Bitmap Wizards

Bitmap wizards are bitmap images, such as windows icons, or a block of text. As you create your application, you can modify picture bitmap wizard images by doing the following:

- Assign an animation to a bitmap
- Flip a image horizontally or vertically
- Change the horizontal and vertical length of a bitmap
- Place a border or a shadow around the bitmap border
- Rotate the bitmap image on its axis in 90 degree increments
- Define a transparent color
- Replace up to three colors in the bitmap with other colors

Texture Wizards

A texture wizard is similar to a bitmap wizard, except that it can be resized to form a continuous pattern. Texture wizards are typically used to create backgrounds for windows or as a fill for a graphic object. You select texture wizards from the Symbol Factory Textures category.

InTouch Object

An InTouch object is an InTouch cell or wizard that is stored "as is" in the Symbol Factory .

After you paste an InTouch object from the Symbol Factory into WindowMaker, you cannot edit it in Symbol Factory .

When you double-click the object in WindowMaker, the **Substitute Tagnames** dialog box appears if the object is a cell, or the animation link selection dialog box appears for an individual graphic object.

Using Symbol Factory

Using the Symbol Factory wizard is very similar to using other wizards. Select a wizard, place it in a window, and configure it.

Getting Started Quickly


If you are familiar with Symbol Factory , review these tips for getting started quickly:

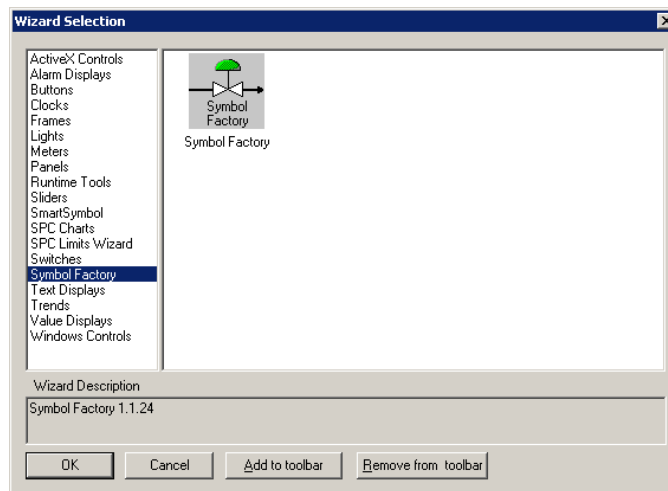
- To configure options for a wizard, double-click it in the window and then click **Options** in the Symbol Factory **by Reichard Software** dialog box.
- To copy a wizard to another category, drag its thumbnail image into the **Categories** window and drop it on another category. To move, hold down the SHIFT key.
- To edit a wizard's description, right-click the wizard thumbnail. To edit a category's description, right-click the category.
- To delete a wizard, right-click the wizard thumbnail with and then click **Delete Symbol**.
- The Symbol Factory can be configured so that a group of developers can use and contribute to the same library of wizards across a network.
- All of the wizards in a particular category are stored in a file with the .cat file name extension. Symbol Factory category files are normally placed in the c:\program files\wonderware\intouch\symfac folder. You can copy the file into the c:\program files\wonderware\intouch\symfac folder on another computer.

Placing a Symbol Factory Wizard in a Window

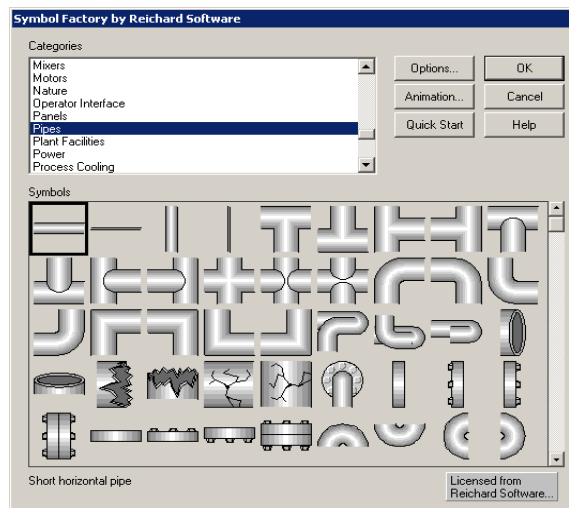
You place a Symbol Factory wizard in a window similar to placing other wizards.

To place a Symbol Factory wizard into a window

- 1 Open an application in WindowMaker.
- 2  Click the Wizard icon in the Wizards/ActiveX Toolbar. The **Wizard Selection** dialog box appears.



- 3 In the list of wizards shown in the left pane, click Symbol Factory .
- 4 Select the Symbol Factory wizard in the display area and then click **OK**.
- 5 Click in the window to place the wizard. The Symbol Factory **by Reichard Software** dialog box appears.



- 6 In the **Categories** list, select a category. The **Symbol** window shows the wizards for the category you selected.
- 7 Select the wizard to place and then click **OK**.

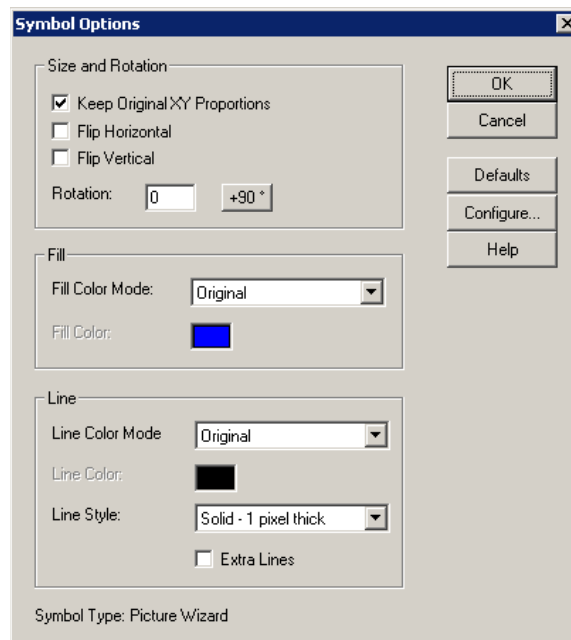
Configuring Symbol Options

Wizard options vary for different wizards. Changing colors will impact the drawing speed of the bitmaps and textures since each pixel must be scanned and possibly changed.

To configure wizard options

- 1 Open a window containing a Symbol Factory wizard.
- 2 Double-click the wizard. The Symbol Factory **by Reichard Software** dialog box appears.
- 3 Keep the wizard selected and click **Options**. The **Symbol Options** dialog box appears.

The image properties shown on the **Symbol Options** dialog box vary by the type of wizard you selected to edit. The following example shows the options that are available when you selected a picture wizard symbol.



Tip If the **Enable alternatives to right mouse button** option is selected in the **Configure Symbol Factory** dialog box, the **Edit Symbol** button is shown and you can click it to configure the selected wizard.

- 4 In the **Size and Rotation** area, do any of the following:
 - Select the **Keep Original XY Proportions** check box to retain the original aspect ratio of the wizard.
 - Select the **Flip Horizontal** check box to flip the wizard horizontally.
 - Select the **Flip Vertical** check box to flip the wizard vertically.
 - In the **Rotation Type** box, type the number of degrees to rotate a wizard. Picture wizards can be rotated to any angle. Bitmap and texture wizards can only be rotated in 90 degree increments (0, 90, 180, or 270). Click the button to automatically increment the rotation angle by 90 degrees.
- 5 If you are configuring a picture wizard, in the **Line** and **Fill** areas, do any the following:
 - In the **Fill Color Mode** list, click a fill type. Double-click the **Fill Color** box to open the color palette.
 - In the **Line Color Mode** list, click a line color. Double-click the **Line Color** box to access the color palette.
 - In the **Line Style** list, click a line style.
 - Select the **Extra Lines** check box to add lines at the borders of gradients within the wizard.
- 6 If you are configuring a bitmap or texture wizard, in the **Effects** and **Change Colors** areas, do any the following:
 - Select the **Include Border** check box to create a black border around a wizard.
 - Select the **Include Shadow** check box to create a dark gray shadow behind the wizard.
 - Click each color box to open the color palette to change the colors in the wizard.
- 7 Click **OK**.

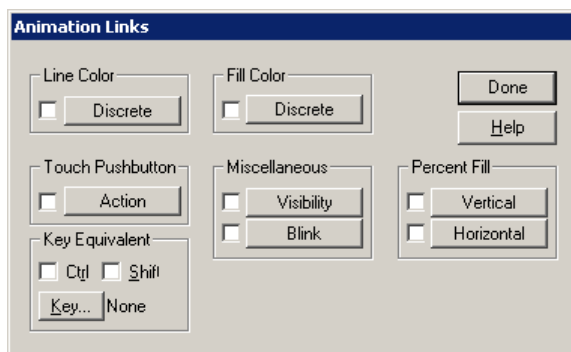
Animating a Wizard

You can animate any Symbol Factory wizard. The Symbol Factory provides you with access to the most common animation links.

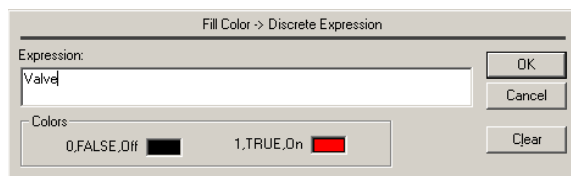
If you need another type of animation link, you must break the wizard and then animate it using the standard InTouch animation links.

To animate a wizard

- 1 Select a wizard in the Symbol Factory, or double-click the wizard if you have already pasted it into your window. The Symbol Factory by Reichard Software dialog box appears.
- 2 Click **Animation**. The **Animation Links** dialog box appears.



- 3 Click the button for each type of animation link to apply to the selected wizard. An expression dialog box appears.



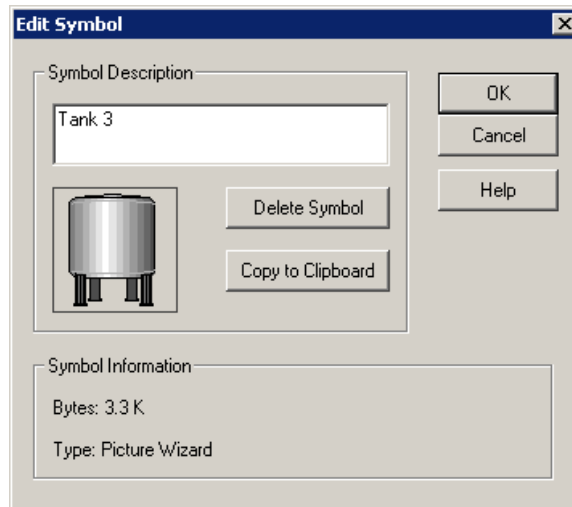
- 4 In the **Expression** window, type the expression.
Double-click in the window to open the **Select Tag** dialog box. If you use an existing tag, you can double-click the tag in your expression to open the **Tagname Dictionary** and see the tag definition.
If you use an undefined tag, you are prompted to define it when you close the expression dialog box.
- 5 Configure the details for the type of animation link.
- 6 Click **OK**.

Editing a Symbol

You can change the description in a wizard tool tip, delete a wizard, or copy a wizard to the Windows clipboard.

To edit a wizard

- 1 In the Symbol Factory **by Reichard Software** dialog box, select the category containing the wizard.
- 2 Right-click the wizard. The **Edit Symbol** dialog box appears.



- 3 Edit the wizard. Do any of the following:
 - In the **Symbol Description** box, type the tool tip text. The maximum description is 80 characters.
 - Click **Delete Symbol** to delete the wizard.
 - Click **Copy to Clipboard** to copy the wizard to the Windows clipboard. If the wizard is a picture wizard, it will be copied as a Windows metafile. If the wizard is a bitmap wizard or texture wizard, it will be copied as a Windows bitmap.
- 4 Click **OK**.

Breaking a Wizard for Editing

You can break a Symbol Factory wizard for individual editing. However, after you break a wizard, it loses its wizard properties. If you accidentally break a wizard, you can reassemble it by using the Undo tool.

To break a wizard

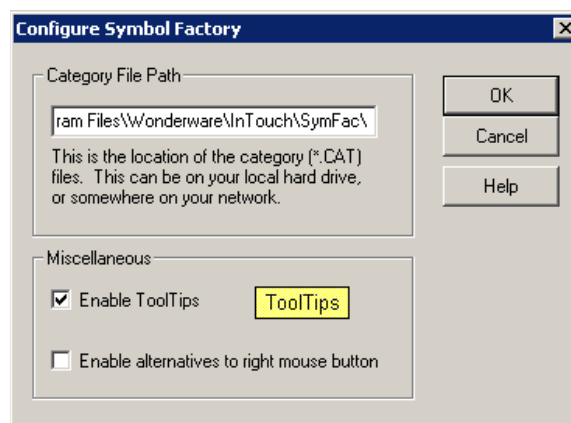
- ◆ On the WindowMaker **Arrange** menu, click **Break Cell**.

Sharing a Category of Symbols on a Network

You can configure the Symbol Factory to allow multiple developers across a network to use and contribute to the category file of wizards.

To move the category file to a network folder

- 1 In the Symbol Factory dialog box, click **Options**. The **Symbol Options** dialog box appears.
- 2 Click **Configure**. The **Configure Symbol Factory** dialog box appears.



- 3 In the **Category File Path** box, type the full path to the network folder where the category file is saved.
- 4 Click **OK**.

Making a Category Read-Only

When storing a wizard file on a network folder, you may want to make the category read-only to prevent other users from moving or renaming wizards.

To make a category read-only

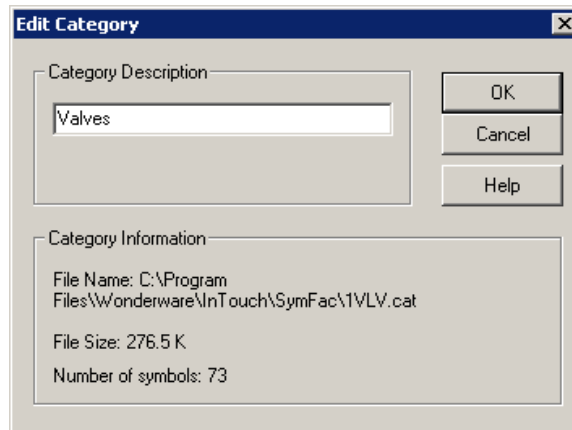
- ◆ Set the file as read-only in Windows Explorer.

Viewing Category Properties

You can view the category path, file size, and number of wizards.

To view properties of a category

- 1 In the Symbol Factory dialog box, right-click the category in the **Categories** list. The **Edit Category** dialog box appears.



- 2 In the **Category Description** box, type the new description for the category, and click **OK**. The maximum length of the description is 40 characters.
- 3 In the **Category Information** area, view the properties.

Category Information	Description
Filename	The category (.cat) file path. By default, this path is c:\program files\wonderware\intouch\ symfac.
File Size	Size of the category file, in kilobytes.
Number of Symbols	Total number of wizards contained in the category. Maximum is 32,767.

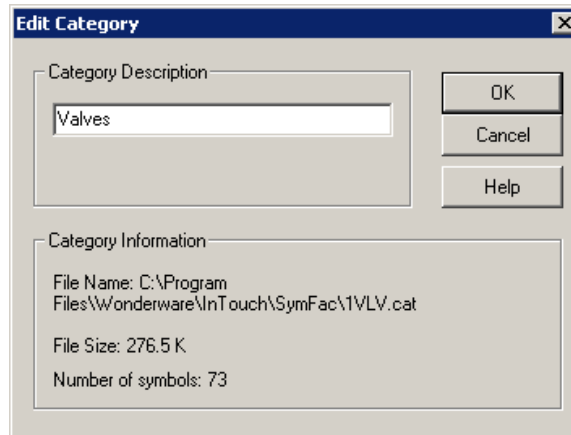
- 4 Click **OK**.

Editing an Existing Category

You can only edit the category name.

To edit an existing category

- 1 In the Symbol Factory dialog box, right-click the category in the **Categories** list. The **Edit Category** dialog box appears.



- 2 In the **Category Description** box, type the new description for the category, and click **OK**. The maximum length of the description is 40 characters.
- 3 Click **OK**.

Deleting a Category

Use Windows Explorer and delete the category (.cat) file by specifying the filename for the category.

Tip You can verify the category filename in the Edit Category dialog box.

Configuring Symbol Factory

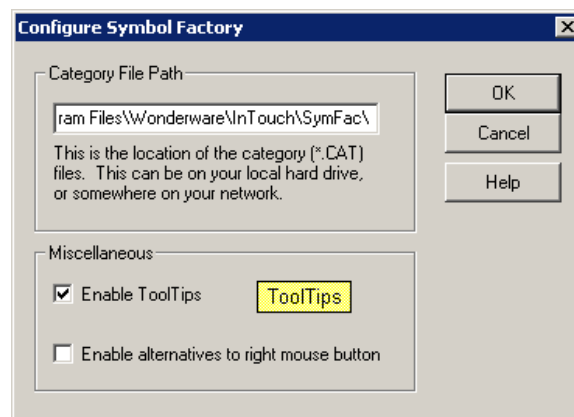
When you configure the Symbol Factory , you can specify:

- Whether tool tips are shown when you select a wizard.
- Whether additional options should appear in the Symbol Factory **by Reichard Software** dialog box. By default, to edit category and wizard descriptions, you must right-click the item.
- The location for your category (.cat) files. All data for all wizards in each category is stored in one category file. For performance reasons, this path should contain only .cat files. To share wizards with other developers, set this path to a network folder. See Sharing a Category of Symbols on a Network on page 217.

Caution Do not place category files in your local InTouch application folder. Instead, save category files to:
C:\Program Files\Wonderware\intouch\symfac

To configure Symbol Factory

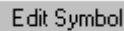
- 1 In the Symbol Factory dialog box, click **Options**. The **Symbol Options** dialog box appears.
- 2 Click **Configure**. The **Configure Symbol Factory** dialog box appears.



- 3 Configure Symbol Factory . Do the following:
 - In the **Category File Path** box, type the location where you want to save your Symbol Factory category (.cat) files.
 - Select the **Enable ToolTips** check box if you want tool tips to be shown for wizards in the Symbol Factory **by Reichard Software** dialog box.
 - Select the **Enable alternatives to right mouse button** check box if you want buttons added to the Symbol Factory **by Reichard Software** dialog box. You can use these buttons instead of the right mouse button for editing categories and wizards:



Shows the **Edit Category** dialog box for a selected category.



Shows the **Edit Symbol** dialog box for a selected wizard.

- 4 Click **OK**.

Troubleshooting

If you accidentally uninstall the Symbol Factory wizard, you need to install it again. For more information on installing wizards, see Chapter 5, Wizards, in *InTouch® HMI Visualization Guide*.

If you accidentally delete a wizard and you want it back, you must retrieve it.

To retrieve a deleted wizard from a category

- 1 Rename the file ~cat.bak to temp.cat.
- 2 Run Symbol Factory and see if the deleted wizard is back. Move it to its original category, then delete the temp.cat file.
- 3 If the above step did not work, hold down the CTRL key while you right-click the category with the deleted wizard. This compacts the category file and creates a fresh backup ~cat.bak.

Perform the previous steps until you find your deleted wizard.

Index

Numerics

16-Pen Trend

- changing configuration at run time 195
- configuring tags to display on graph 191–192
- configuring trend display options 194
- configuring trend time span and update rate 193
- controlling with scripts 196–207
- creating 190
- description 189

D

- documentation conventions 9

F

functions

- ptGetTrendType() function 196
- ptLoadTrendCfg() function 197
- ptPanCurrentPen() function 198
- ptPanTime() function 199
- ptPauseTrend() function 200
- ptSaveTrendCfg() function 201
- ptSetCurrentPen() function 201
- ptSetPen() function 202
- ptSetPenEx() function 203
- ptSetTimeAxis() function 204
- ptSetTimeAxisToCurrent() function 205
- ptSetTrend() function 205
- ptSetTrendType() function 206
- ptZoomCurrentPen() function 206
- ptZoomTime() function 207

- RecipeDelete() function 109
- RecipeGetMessage() function 116
- RecipeLoad() function 107
- RecipeSave() function 108
- RecipeSelectNextRecipe() function 112
- RecipeSelectPreviousRecipe() function 113
- RecipeSelectRecipe() function 111
- RecipeSelectUnit() function 110
- SPCConnect() function 25
- SPCCreateNewProduct() function 79
- SPCDatasetDlg() function 65
- SPCDeleteProduct() function 79
- SPCDisconnect() function 26
- SPCDisplayData() function 70
- SPCLocateScooter() function 71
- SPCMoveScooter() function 71
- SPCSampleDeleted() function 75
- SPCSaveSample() function 74
- SPCSelectDataset() function 67
- SPCSelectProduct() function 68
- SPCSetControlLimits() function 80
- SPCSetMeasurement() function 73
- SPCSetProductCollected() function 77
- SPCSetProductControlLimits() function 82
- SPCSetProductDisplayed() function 68
- SPCSetProductRangeLimits() function 83
- SPCSetProductSpecLimits() function 84
- SPCSetRangeLimits() function 81
- SPCSetSpecLimits() function 81

SQLAppendStatement() 159
 SQLClearParam() function 173
 SQLClearStatement() function 176
 SQLClearTable() function 156
 SQLCommit() function 178
 SQLConnect() function 138
 SQLCreateTable() function 139
 SQLDelete() function 157
 SQLDisconnect() function 139
 SQLDropTable() function 140
 SQLEnd() function 148
 SQLErrorMsg() function 181
 SQLExecute() function 174
 SQLFirst() function 146
 SQLGetRecord() function 145
 SQLInsert() function 149
 SQLInsertEnd() function 152
 SQLInsertExecute() function 151
 SQLInsertPrepare() function 150
 SQLLast() function 148
 SQLLoadStatement() function 160
 SQLManageDSN() function 180
 SQLNext() function 147
 SQLNumRows() function 146
 SQLPrepareStatement() function 162
 SQLPrev() function 147
 SQLRollback() function 178
 SQLSelect() function 142
 SQLSetParamChar() function 163
 SQLSetParamDate() function 164
 SQLSetParamDateTime() function 165
 SQLSetParamDecimal() function 166
 SQLSetParamFloat() function 167
 SQLSetParamInt() function 168
 SQLSetParamLong() function 169
 SQLSetParamNull() function 170
 SQLSetParamTime() function 172
 SQLSetStatement() function 158
 SQLTransact() function 177
 SQLUpdate() function 153
 SQLUpdateCurrent() function 155

P

ptGetTrendType() function 196
 ptLoadTrendCfg() function 197
 ptPanCurrentPen() function 198
 ptPanTime() function 199
 ptPauseTrend() function 200

ptSaveTrendCfg() function 201
 ptSetCurrentPen() function 201
 ptSetPen() function 202
 ptSetPenEx() function 203
 ptSetTimeAxis() function 204
 ptSetTimeAxisToCurrent() function 205
 ptSetTrend() function 205
 ptSetTrendType() function 206
 ptZoomCurrentPen() function 206
 ptZoomTime() function 207

R

Recipe Manager

- clearing data from a range of cells 96
- configuring the editing grid 93–94
- copying a range of cells 96
- defining ingredient names and data types 99
- defining ingredient values 101
- deleting a recipe template file 102
- deleting a template column 98
- deleting a template row 98
- description 89–90
- editing a recipe template file 102
- editing recipe data 93
- inserting a column in a template 98
- inserting a row in the Template Definition template 97
- mapping InTouch tags to ingredients 100
- nesting recipes 105
- template files 92
- user interface 91
- using Excel to edit a recipe template file 103
- using Notepad to edit a recipe template file 104
- using recipes in InTouch applications 106
- working with editing grid 95

RecipeDelete() function 109
 RecipeGetMessage() function 116
 RecipeLoad() function 107
 RecipeSave() function 108
 RecipeSelectNextRecipe() function 112
 RecipeSelectPreviousRecipe() function 113
 RecipeSelectRecipe() function 111
 RecipeSelectUnit() function 110

S

- SPCConnect() function 25
- SPCCreateNewProduct() function 79
- SPCDatasetDlg() function 65
- SPCDeleteProduct() function 79
- SPCDisconnect() function 26
- SPCDisplayData() function 70
- SPCLocateScooter() function 71
- SPCMoveScooter() function 71
- SPCPro
 - changing the currently collected product 76–77
 - changing the data collection stagger time 27
 - changing the dataset used by a chart 66–67
 - changing the displayed product in a chart 67–68
 - collecting SPC data automatically 23
 - configuring a product for a dataset 20
 - configuring a SQL Server database 15
 - configuring Access database 13
 - configuring collection and analysis options 17
 - configuring SPC database users 24
 - configuring the database 12
 - creating new products during run time 78
 - description 11
 - entering a new sample 73–74
 - entering calculation parameters 80–84
 - entering data into attribute charts 85
 - manually collecting data with chart wizards 28
 - migrating SPC databases from a previous SPCPro version 86–88
 - scrolling a chart 69–70
 - updating a chart to the current time 72
- SPCSampleDeleted() function 75
- SPCSaveSample() function 74
- SPCSelectDataset() function 67
- SPCSelectProduct() function 68
- SPCSetControlLimits() function 80
- SPCSetMeasurement() function 73
- SPCSetProductCollected() function 77
- SPCSetProductControlLimits() function 82
- SPCSetProductDisplayed() function 68
- SPCSetProductRangeLimits() function 83
- SPCSetProductSpecLimits() function 84
- SPCSetRangeLimits() function 81
- SPCSetSpecLimits() function 81
- SQL Access Manager
 - clearing statement parameters 173
 - connecting and disconnecting the database 138–139
 - creating a new table 139–140
 - creating a statement or loading a statement from a file 160–161
 - defining structure of a new table 126–127
 - deleting a table 140
 - deleting records from a table 156–157
 - description 119–120
 - executing a statement 174–176
 - executing parameterized statements 158–159
 - mapping InTouch tags to database columns 122–125
 - opening the ODB administrator dialog box at run time 180
 - performing common SQL operations in InTouch 132–137
 - preparing a statement 161–162
 - releasing occupied resources 176
 - reserved word list 185–187
 - retrieving data from a table 141–148
 - setting statement parameters 163–172
 - setting up an ODBC source 121
 - understanding SQL error messages 181–184
 - updating existing records in a table 153–155
 - working with Microsoft Access applications 130
 - working with Oracle applications 131–132
 - working with SQL Server applications 128–129
 - working with transaction sets 177–179
 - writing new records to a table 149–152
- SQLAppendStatement() function 159
- SQLClearParam() function 173
- SQLClearStatement() function 176
- SQLClearTable() function 156
- SQLCommit() function 178
- SQLConnect() function 138

SQLCreateTable() function 139
SQLDelete() function 157
SQLDisconnect() function 139
SQLDropTable() function 140
SQLEnd() function 148
SQLErrorMsg() function 181
SQLExecute() function 174
SQLFirst() function 146
SQLGetRecord() function 145
SQLInsert() function 149
SQLInsertEnd() function 152
SQLInsertExecute() function 151
SQLInsertPrepare() function 150
SQLLast() function 148
SQLLoadStatement() function 160
SQLManageDSN() function 180
SQLNext() function 147
SQLNumRows() function 146
SQLPrepareStatement() function 162
SQLPrev() function 147
SQLRollback() functions 178
SQLSelect() function 142
SQLSetParamChar() function 163
SQLSetParamDate() function 164
SQLSetParamDateTime() function 165
SQLSetParamDecimal() function 166

SQLSetParamFloat() function 167
SQLSetParamInt() function 168
SQLSetParamLong() function 169
SQLSetParamNull() function 170
SQLSetParamTime() function 172
SQLSetStatement() function 158
SQLTransact() function 177
SQLUpdate() function 153
SQLUpdateCurrent() function 155
Symbol Factory

- animating a wizard 215
- breaking a wizard for editing 217
- configuring symbol options 213–214
- editing a symbol 216
- getting started quickly 211
- making a category read-only 218
- placing a wizard in a window 212–213
- sharing a category of symbols on a network 217
- symbol types 209–211
- troubleshooting 221
- viewing category properties 218

T

technical support, contacting 10
troubleshooting, Symbol Factory 221