Infor Mongoose Software Development User Guide

Table of Contents

Application Messages

About Application Message Construction

Use application messages to display these types of information:

- Information or questions that require responses from the user
- Warnings about potential problems or conditions
- Error conditions within the Mongoose environment or operations
- System responses to user actions

NOTE: The types of application messages described here are generated through application database code (stored procedures, triggers, etc.) and/or in the IDO layer (IDO custom assemblies, etc.). Messages that are generated at the client are not applicable here and do not reference or use the messages that are contained in these Application tables.

Application messages are stored in the system and are invoked in response to user or system operations. Messages can be constructed with both literal strings and variable values substituted and inserted into the message when it is invoked.

Messages are identified and invoked with a unique message number that is assigned to the message when it is constructed.

Setting Message Numbers

Typically, the unique message number consists of an alphanumeric prefix that identifies the application or its owner, followed by an autonumbered suffix that is created and applied when the message is constructed. You are not required to follow this model, but it provides the best way to identify and locate messages.

For example, your organization is called WonderWare, and you have an application called IssueTrack. You can create and designate a message number prefix like "WW-IssTr-" to identify all messages used in conjunction with this application. If you add an autonumbering suffix to this, you can then create messages for your application without worrying about maintaining unique message numbers. Using this process also allows you to locate all messages designed for use with this application.

To create this type of message numbering:

1. Use the **Message Num Prefix** field on the **System Parameters** form to specify the prefix. The system uses the prefix automatically when messages are created.
2. Use the **Maintain Application Messages** form to construct the message.

For information about constructing messages, see the next section.

The **Message Num** column on this form is defined as a TBD (To Be Determined) field, which means that, when you create a message and save it, the system automatically uses the prefix and assigns the next available number as the suffix.

3 Continuing our example, when you create your first application message, the system automatically assigns the message number "WW-IssTr-1" to it. The next message is created as "WW-IssTr-2" and so on.

Note: Predefined core messages assigned to and used by Mongoose have the prefix "MG_". Other Mongoose-based applications might have and use other prefixes; for example, "SL_" for SyteLine-related messages, "SM_" for Service Management-related messages, and so on.

4 Save the message.

Constructing Messages

A message can be constructed in such a way that values of various parameters can be picked up and used dynamically when the message is invoked. This is done with the use of substitution parameters that are evaluated when the message is invoked and run-time values substituted for the parameters.

These substitution parameters are indicated in a message by the use of an ampersand (&) followed by a number or letter. Examples: **&5** and **&C**

Possible sources of values for these substitution parameters include:

- Row captions
- Column or property captions
- Column or property values (literal or variable)
- Column or property value-captions (that is, the translatable caption that appears in the drop-down list of one or more combo box components on a form)

When constructing a message with substitution parameters, make sure the code that invokes the message contains the correct number of values for the substitutions.

To invoke the message at run-time, insert an expression into your code at the point where you want to call the message. This expression must use the MESSAGE parameter function.

Example: The message **E=NoExistForIs5** has the value in English: "There exists no &1 where &2 is &3 for &4 that has [&5: &6] and [&7: &8] and [&9: &A] and [&B: &C] and [&D: &E]."

This message requires 14 parameters as follows:

- &1 Row caption
- &2 Column or property caption
- &3 Column or property value
- &4 Row caption
- &5 Column or property caption
- &6 Column or property value

- &7 Column or property caption
- &8 Column or property value
- &9 Column or property caption
- &A Column or property value
- &B Column or property caption
- &C Column or property value
- &D Column or property caption
- &E Column or property value

An example of a call for this message might look like this:

```
MESSAGE("E=NoExistforIs5", "@serial", "@serial.ser_num", V(SerNum)
    , "@item"
    , "@serial.whse", V(Whse)
    , "@serial.item", V(Item)
    , "@serial.loc", V(Loc)
    , "@serial.lot", V(Lot)
    , "@rsvd_inv.import_doc_id", V(ImportDocId)
    )
```

At action execution time, this might evaluate to the following string:

There exists no Serial Number where S/N is S/N1234 for Item that has [Whse: MAIN] and [Item: BK-27000-0007] and [Location: STOCK] and [Lot: LOT00012345] and [Import Doc Id: DocId000123456].

Invoking and Concatenating Multiple Messages

You can construct calls to display multiple messages simultaneously. To do this, use multiple MESSAGE expressions separated by pipe (|) symbols. The pipe symbol concatenates the messages.

Example: Consider the following MESSAGE expressions:

```
MESSAGE("E=NoExistforIs5", "@serial", "@serial.ser_num", V(SerNum)
    , "@item"
    , "@serial.whse", V(Whse)
    , "@serial.item", V(Item)
    , "@serial.loc", V(Loc)
    , "@serial.lot", V(Lot)
```

```
        , "@rsvd_inv.import_doc_id", V(ImportDocId)
        )
       | MESSAGE("E=AppLockFail3")
```

At run time, this concatenation might evaluate to this message conversation:

There exists no Serial Number where S/N is S/N1234 for Item that has [Whse: MAIN] and [Item: BK-27000-0007] and [Location: STOCK] and [Lot: LOT00012345] and [Import Doc Id: DocId000123456].

Lock request was chosen as a deadlock victim.

Building Constraint Exception Messages

When a SQL constraint exception is thrown from the application database, the IDO Request layer catches the exception and can build a translatable message from the SQL constraint name and type, if a message for that constraint name exists in the ObjectMainMessages table. Different constraints can use the same basic message, which varies only by the different object names that are referenced in the message text. For example, many constraint exceptions could be reported to a user with this basic message:

The &1 entered already exists

where the **&1** substitution expression could refer to any one of hundreds of different objects.

However, the text (or the object name that references the text) to be substituted cannot be passed when the constraint exception occurs. The IDO Request layer can only pick up the constraint name and type from the caught exception. The ObjectBuildMessages table contains child records that reference either a Message Number defined on the **Maintain Application Messages** form or another Object Name defined on the **Maintain Application Message Objects** form for the text to be used for each substitution expression that exists in the referenced base.

Thus the same base message from the ApplicationMessage table can be used by many different constraints, each of which defines a different set of references for the substitution text placeholders in the message.

Example

The existing message number SL_100001 has the text "The &1 entered is not valid." You can reuse this message number and text to construct a custom constraint error message.

1. Specify an Object Name for a SQL constraint, using the rules specified in the field description.
2. Select an existing Message Text, and the Message Number is displayed. For example, if you select the Message Text "The &1 entered is not valid." the Message Number SL_100001 is displayed.

3 Specify a Message Description that describes how and when this instance of the message is used. For example, "This message is displayed when <state> is updated on an Applicant Reference to a value that does not exist in any state."

4 Specify the Message Type. For any given Object Name, you can only have one Message Type with the same value. For example, select Message Type 17 (Constraint Message).

5 Specify the Object Type. For example, select 0 (Table Object).

6 Specify the Message Severity.

7 Save the record. If the type is 17 (Constraint Message) or 18 (Delete Constraint), the Build Messages grid is enabled, where you can specify this information:

- If the Message Text in the main grid has multiple substitution expressions (&1, &2, etc.), add a row with a Sequence Number corresponding to each of the substitution expressions.
- To reference the substitution text to use, either specify a Message Number from the **Maintain Application Messages** form or specify another existing Object Name.
- A Message Number is always required. Specify **MG_1** to indicate that the value in the Object Name field is to be used to look up the substitution text. If the Message Number is any value other than MG_1, then that value is used to look up the message text on the **Maintain Application Messages** form.
- If the Object Name field is used, select the appropriate value from the drop-down list.
- The Message Text field displays the text that will be displayed in place of the substitution expression.

8 Save the record again.

9 If your company uses source code control, click **Generate Message Script File**. In the **Generate Application Messages Script** form, specify the appropriate file path and filter information, and click **Generate SQL Script File**.

IDOs

Understanding IDOs

An Intelligent Data Object (IDO) is a business object that encapsulates units of information and logic that are called from the client layer to interact with data in the database. The job of the IDO is to transport collections of data back and forth, with any validation or rules needed, between the client and the database.

An IDO consists of these elements:

- **A set of one or more SQL tables.** Each table contains the data for a specified part of the application and must include columns (properties) that Mongoose requires to work properly.
- **A set of properties.** A property may represent persistent data stored in the application database, derived data, or temporary data used to communicate information to the middle tier. A property may also represent a whole subcollection of data.
- **A set of standard methods.** All IDOs implement the methods **LoadCollection**, **UpdateCollection**, **GetPropertyInfo**, and **Invoke**:
    - **LoadCollection** retrieves a collection of rows from the database.
    - **UpdateCollection** takes a set of rows marked for insert, update, or delete, and executes the appropriate SQL code against the database.
    - **GetPropertyInfo** returns detailed information about the properties supported by the IDO.
    - **Invoke** allows you to execute a custom method.

Through configurations, application databases are linked with an objects database and a forms database.

IDO forms serve as a development environment for IDOs. IDO definitions are stored as metadata in the objects database. You can edit the metadata through the IDO forms.

Forms use IDOs in multiple ways. Forms that interact with the application database data define collections based on IDOs. Many types of validators and list sources are built over IDOs.

The Application Event System (AES) leverages IDOs. Many of the framework AES events are generated as the application operates on IDOs. Many of the actions provided in AES operate on IDOs, allowing you to quickly define business processes, automation rules, or general server-side logic in your application.

Extend and Replace IDOs

The IDO development system allows a developer to create a new IDO that inherits all the properties, methods, tables, and the extension class from an existing IDO. The relationship between the created IDO and the base IDO is called an "extends" relationship; the new IDO inherits from the base IDO and extends it.

The developer can optionally flag an extending IDO as a replacement for the base IDO. When an extending IDO replaces its base IDO, all IDO requests that are targeted for the base IDO are rerouted through the extending IDO. These requests are the **GetPropertyInfo**, **LoadCollection**, **UpdateCollection**, and **Invoke** requests.

The options **Extends** and **Extend and Replace** are available in the **New IDO Wizard**, used to add an IDO.

An extending IDO (one with a base IDO) can only make additive changes to the base IDO. That is, new properties, methods, tables, and an extension class can be added, but none of the base properties, methods, tables, or extension classes can be modified or deleted. New bound properties can be bound to columns from tables in the extending IDO or in any base IDO. Likewise, new derived properties can reference properties in the extending IDO as well as properties in any base IDO.

An extending IDO may itself be extended by one or more other IDOs. There is no hard limit to the number of levels in an inheritance chain. While any number of extending IDOs can share the same base IDO, no more than one sibling IDO can be flagged as a replacement for the base IDO.

When the IDO runtime processes IDO requests, it generates events that may be handled by IDO extension classes. If an extending IDO has one or more base IDOs that have extension classes associated with them, the events will fire in all extension classes, but the order is indeterminate.

The base IDO and the replace flag can only be set when creating a new IDO. The attributes are read-only for existing IDOs.

Working with IDO Projects

Adding an IDO Project

An IDO project is a group of one or more IDOs.

To add an IDO project:

1. Open the **IDO Projects** form.
2. Select **Actions>New** or click the "Create a New Object" button on the toolbar.
3. Enter the name of your project. The name must:
   - Be unique in the Objects database
   - Consist of alphanumeric characters (no spaces)
   - Begin with a letter

- Be no longer than 30 characters.

4 Select **Actions>Save** or click the save button on the toolbar.

Deleting IDO Projects

You can delete any project in the current Objects database. Deleting a project is not allowed if any IDOs are attached to the project.

To delete an IDO project:

1 In the **IDO Projects** form, select the desired project.

2 Select **Actions>Delete** or click the **Delete** button on the toolbar.

3 Select **Actions>Save** or click the **Save** button on the toolbar.

Working with IDOs

The Basics

Viewing and understanding an IDO definition

To view the elements and attributes of an IDO:

1 Open the IDOs form.

2 In the grid view, select the IDO you want to view.

When viewing the elements and attributes of an IDO keep these particulars in mind:

This field or group of fields:	Does or tells you this:
Attributes group	The basic and identifying attributes for the IDO, which include: • The name of the IDO • The IDO project with which it is associated • Any IDO-level properties or associated property classes • Other advanced attributes the IDO might have For more information about specific attributes and elements in this group, use the context-sensitive help for the specific attribute or element (Right-click > Help).
Group for source	Revision status and check-out status of the IDO

control status

Buttons for modifications

The row of buttons allow you to access a number of different forms from which you can gain additional information about elements of the IDO, add elements to the IDO, and make other modifications to the IDO.

This button:	Opens this form:	Which allows you to:
Tables	IDO Tables	View, add, and edit tables in the IDO
Properties	IDO Properties	View, add, and edit properties for a designated table in the IDO
Methods	IDO Methods	View, specify, and edit methods and stored procedures used within an IDO
Filters	Row Authorizations	View, specify, and edit criteria by which the data from the IDO collection is filtered
New IDO	New IDO Wizard	Add an IDO to the associated IDO project
New Table	New Table	Define a base table for an IDO Add or remove a secondary table Define joins between the tables used in an IDO

	New Property	New Property	Add a property to a designated table
	New Method	New Method	Define a new method to be used with the IDO

Status subcollection notebook	In summary form, the elements and rules associated with the selected IDO:

This tab:	Displays a list and condensed view of:
Tables	All the tables associated with the IDO
Properties	All the properties defined in the IDO
Methods	All the methods associated with the IDO
Filters	The filters applied to the IDO, whether active or not
Rules	The rules that define relationships between IDOs and the policies to enforce when records are deleted

About Source Control

You can use a source control system to allow multiple developers in the system the ability to customize and maintain forms without having to worry about overwriting one another's changes.

Mongoose now provides support for multiple source control options:

- Microsoft Visual SourceSafe
- Microsoft Team Foundation Server
- Apache Subversion

If you plan to use any of these source control systems to manage changes to forms and other system components, you must:

- Obtain, install, and configure the source control software.

- Use the Configuration Manager to configure the source control software to work with your system. For more information, see the *Installation Guide* and the online Help for the Configuration Manager.
- Configure the user profile for each system user who is authorized to check forms and other components in and out. For more information, see Users (**Source Control** tab).

NOTE: This is a change from some previous versions, where Mongoose itself was configured to use the source control system, and not individual users.

About Non-Mongoose Data Used in Mongoose Applications

You can incorporate data from a non-Mongoose source, for example a legacy application that is being converted to a Mongoose base. However, certain schema elements required by Mongoose often do not exist in the non-Mongoose data source, as described in this topic.

To incorporate these data sources into your Mongoose-based application, you must create views that provide the schema elements required by Mongoose. For more information, see the appropriate topic:

- Including Data from a Different SQL Database into a Mongoose Application
- Including Data from an Oracle Database into a Mongoose Application

NOTE: Currently, only SQL Server or Oracle data sources can be linked to Mongoose applications.

Unicode Support

Mongoose databases are designed to support Unicode, but other data sources might not support Unicode. To avoid improper scanning of indexes in the non-Mongoose data sources, you must set a process default in Mongoose. See the entry for **Non Unicode Literal** in the "Default Name" help topic.

Optimistic Locking

Mongoose databases use the RecordDate property to provide optimistic locking. In many cases, however, the non-Mongoose data source does not have a RecordDate column. So, one of the major problems with using non-Mongoose data sources is the coordination of optimistic locking. Both the **IDO Linked Databases** form and the **IDO Linked Tables** form provide options to specify a column to be used for optimistic locking in the non-Mongoose data source.

These options include:

- Allowing Mongoose to identify whether a "RecordDate" column exists in the non-Mongoose data source
- Designating a default column on the **IDO Linked Databases** This is the name of a column that is normally used for optimistic locking by the non-Mongoose tables, if such a column exists.
- Designating a column for optimistic locking on the **IDO Linked Tables** form

If none of these options exist, the literal string "NODATE" is designated as the "RecordDate" value. Because this literal string is applied to any access of the non-Mongoose data source, no optimistic locking occurs.

About the Mongoose View

The created Mongoose view over the non-Mongoose data source includes the columns from the data source plus these additional columns, required by Mongoose for processing:

- "RecordDate", used for optimistic locking in Mongoose. For a non-Mongoose SQL data source, the system assigns whatever column you have assigned to use for optimistic locking to this value. During run time, Mongoose checks to see if this value has been modified elsewhere since the data was first queried, before your modifications. For an Oracle data source, this value is derived.
- "RowPointer", required to be a value that is unique for the entire table. For an Oracle data source, this value is derived.
- "AddMongooseFields", with a literal value of **1**, which is used internally.

After this view is created, you can create IDOs and forms, and perform read and write operations on the linked database table like any other Mongoose database table. One exception is that you cannot use the non-Mongoose database and tables for any event action where the workflow must be suspended. See the *Guide to the Application Event System*.

Including Data from a Different SQL Database into a Mongoose Application

To incorporate data from a non-Mongoose linked SQL Server database into a Mongoose-based application, you must specify information about the other database in the **IDO Linked Database** form and the **IDO Linked Tables** form.

NOTE: If you want to link to a non-Unicode database, use the **Process Defaults** form to set the process default for **Non Unicode Literal**. This helps ensure that the database indexes are scanned and accessed properly when performing queries. See the entry for **Non Unicode Literal** in the "Default Name" help topic.

To incorporate data from a non-Mongoose SQL database for use in a Mongoose-based application:

1 Open the **IDO Linked Database** form and specify these values:

Link Database

Specify the name that is to identify the database in Mongoose. This is a Mongoose internal designation only and need not be the same as the actual database name.

Database Name

Specify the SQL Server name of the database to which you want to link. If this database resides in the same location as the Mongoose databases, you can provide just the name of the database. If this database resides in a location other than the Mongoose databases, you must also provide the location of the database. Use this format: databaseServer.databaseName

Optimistic Lock Column Name

Optionally, specify the name of a column that might be available for optimistic locking in the non-Mongoose database. If no specific column is designated on the IDO Linked Tables form, this value is used as the default optimistic lock setting.

2 Save your changes.

3 Open the **IDO Linked Tables** form to verify and adjust the column settings for the non-Mongoose table:

- Verify that the columns listed in the **Column Name** column match those of the non-Mongoose table.
- Optionally, rename the **View Column Names** as you want them to display in your Mongoose application.
- Verify that the primary keys for the non-Mongoose table are correct. Modify the choices for keys as necessary.
- Optionally, select the column to use for optimistic locking in the non-Mongoose database. Mongoose designates the “NODATE” literal string for the "RecordDate" value, and no optimistic locking is performed when these conditions exist:
  - No "RecordDate" column exists in the non-Mongoose table.
  - The column specified in the **Optimistic Lock Column Name** field of the **IDO Linked Databases** form does not exist in the non-Mongoose table.
  - No column is designated for optimistic locking on the **IDO Linked Tables** form.

4 To create the Mongoose view, click **Create View**.

Mongoose creates a view that includes the columns from the linked table, along with columns and values for:

- "RecordDate", used for optimistic locking
- "RowPointer", required to be a value that is unique for the entire table
- "AddMongooseFields", with a literal value of **1**, which is used internally

You can now use the linked database and tables in the same way that you use any database created within Mongoose. You can create IDOs and forms, and perform read-write operations on them like any other Mongoose database. However, you cannot use the non-Mongoose SQL database and tables for any event action where the workflow must be suspended. See the Guide to the Application Event System.

Including Data from an Oracle Database into a Mongoose Application

Mongoose-based applications can communicate with Oracle databases through the IDO layer. To set up communication, use these steps:

NOTE: If you want to link to a non-Unicode database, use the **Process Defaults** form to set the process default for **Non Unicode Literal**. This ensures that the database indexes are scanned and accessed properly when performing queries. See the entry for **Non Unicode Literal** in the "Default Name" help topic.

1 In the **Outrigger Profiles** form, create a profile for the Oracle database. Set the database type to **Oracle**, and specify the Oracle user, password, and data source to use when linking to the database. The data source represents the Oracle service name as defined in the tnsnames.ora file.

2 In the **IDO Linked Databases** form, create a new linked database record with these values:

Link Database

Specify the name that is to identify the database in Mongoose. This is a Mongoose internal designation only and need not be the same as the actual database name.

Database Type

Specify Oracle.

Profile Name

Specify the profile created in step 1.

3 In the Tables grid, specify an existing table name in the Oracle database.

4 Save the table record, which enables the buttons on the form.

5 Select the table and click **Columns**. This brings up the **IDO Linked Tables** form, with a Columns grid.

6 Click **Repopulate** to connect to the Oracle database and create the default column information for your new table.

7 Click **Create View** to create a new Mongoose view that defines the necessary columns and data types.The Mongoose view includes the columns from the linked table, along with columns and values for:

- "RecordDate", used for optimistic locking
- "RowPointer", required to be a value that is unique for the entire table
- "AddMongooseFields", with a literal value of **1**, which is used internally

8 Use the **New IDO Wizard** to create an IDO over your new table. The profile name is automatically populated.

Notes

- The profile stored with an IDO definition is used to make an ApplicationDB object within the IDO layer for read and write operations on the table, as well as for method calls to that database.
- As an alternative to building an IDO through the wizard, which allows basic read-write functionality to the Oracle database, you can use IDORuntime.Context.CreateOutriggerApplicationDB in a custom program to provide direct access to the outrigger database. See Example: Custom Code to Communicate with an Outrigger Database for a code sample that uses this assembly.
- The standard Oracle client DLLs are required on the Mongoose utility server. Non-direct mode is used for communications.
- Asynchronous event handlers cannot be used when communicating with Oracle databases.
- User-defined fields cannot be used when communicating with Oracle databases.
- All Oracle tables referenced in an IDO must exist in the same Oracle database
- RowPointer and RecordDate properties are used by Mongoose but do not exist in Oracle tables, so those IDO properties are derived instead of pointing to base table columns.
- A called Oracle IDO method must be a function with a return type of integer.

Example: Custom Code to Communicate with an Outrigger Database

This code sample uses a custom method IDORuntime.Context.CreateOutriggerApplicationDB to provide direct access to an outrigger database. The database must first be linked to your Mongoose application as described in Including Data from an Oracle Database into a Mongoose Application.

```
      public int IdoLinkOtherDbColPopulateSp( string linkedDatabase, string
tableName, string optimisticColumnName, byte databaseType, string profileName,
string infobar )
      {
         int result = 0;
         if ( databaseType == 1 ) // SQL Server
         {
            if ( !DoSqlServerColumnsSp( linkedDatabase, tableName, infobar ) )
               return 16;
            else
               return 0;
         }

         if (!DeletePreviousColumns( linkedDatabase, tableName, infobar ) )
            return 16;
         using ( ApplicationDB db =
IDORuntime.Context.CreateOutriggerApplicationDB( profileName ) )
         {
            IDbCommand cmd = db.Connection.CreateCommand();
            IDbDataParameter parm;

            cmd.CommandType = CommandType.Text;
            cmd.CommandText = @"
SELECT
  cols.COLUMN_NAME
 , cols.COLUMN_NAME As ViewColumnName
 , CASE WHEN DATA_TYPE IN ('VARCHAR', 'NVARCHAR', 'VARCHAR2', 'NVARCHAR2') THEN 1
ELSE 0 END AS IsCharacterColumn
 , CASE WHEN xx.TABLE_NAME IS NULL THEN 0 ELSE 1 END AS IsKeyColumn
, cols.DATA_TYPE
, CASE WHEN cols.DATA_PRECISION IS NULL THEN cols.DATA_LENGTH ELSE DATA_PRECISION
END
```

```
, cols.DATA_SCALE
FROM user_tab_cols cols
LEFT OUTER JOIN (
select ucc.table_name, ucc.column_name
from user_constraints uc
inner join user_cons_columns ucc on
  ucc.table_name = uc.table_name
and ucc.constraint_name = uc.constraint_name
and uc.constraint_type = 'P'
 ) xx ON
xx.TABLE_NAME = cols.TABLE_NAME
AND xx.COLUMN_NAME = cols.COLUMN_NAME
WHERE cols.TABLE_NAME = :ptable
";
            parm = cmd.CreateParameter();
            parm.ParameterName = "ptable";
            parm.Value = tableName.ToUpper();
            cmd.Parameters.Add( parm );
            cmd.CommandType = CommandType.Text;
            cmd.Connection = db.Connection;
            IDataReader colReader = cmd.ExecuteReader();
            while (colReader.Read() )
            {
               string colName = colReader.GetString(0);
               string viewColName = colReader.GetString(1);
               byte isCharacter = colReader.GetByte(2);
               byte isKey = colReader.GetByte(3);
               byte isOptimisticLock = 0;
               string propertyDataType = colReader.GetString(4);
               int propertyLength = colReader.GetInt32(5);
               int? propertyScale = colReader.GetInt32(6);
               if ( colReader.IsDBNull(6) )
                  propertyScale = null;
               if ( optimisticColumnName == colName )
                  isOptimisticLock = 1;
```

```
                if (!InsertOneColumn( linkedDatabase, tableName, colName,
viewColName, isCharacter, isKey, isOptimisticLock, propertyDataType,
propertyLength, propertyScale, infobar ) )
                {
                   break;
                }
             }
          }
          return result;
       }
```

Checking In and Out

Checking In IDOs

When you finish working with an IDO, you check it into the current objects database. If you use a source control system, checking an IDO into the objects database also checks it into the source control system. (New IDOs do not exist in source control until you check them in.)

To check in the current IDO:

1. Open the **IDOs** form and select the desired IDO.
2. Click **Check In**.
3. You are prompted to enter comments. Check In comments are stored in the source control system.
4. Click **OK**.

Checking Out IDOs

If an IDO has been checked into the current Objects database, you must check it out before you can work with it.

To check out the current IDO:

1. Open the **IDOs** form and select the desired IDO.
2. Select **Check Out**.

Adding, Editing, and Deleting IDOs

Adding an IDO

An IDO (Intelligent Data Object) provides a set of properties and methods, and implements the IDO request interface (LoadCollection, UpdateCollection, and Invoke).

You can add IDOs to an IDO project.

To add an IDO to the current project:

1. From either the **IDO Projects** or **IDOs** form, click **New IDO**.

2 In the first page of the New IDO Wizard, name the IDO and identify the primary base table that stores values for the properties of this IDO.

3 Click **Next**.

4 In the second page of the wizard, create bound properties for this IDO that are based on columns in the primary base table in the application database.

By default, all of the columns in the primary base table are included as properties in the IDO.

- To remove a column and omit it as a property, clear the **Inc** check box for that column.
- To change a single property name, specify the new name in the **Property Name** field.
- To make changes to all property names at one time, click **Massage Property Names**, which opens the Massage Options form.

5 Click **Finish**.

WinStudio displays the new IDO, automatically checked out to you.

Setting Advanced IDO Attributes (Primary Keys)

The advanced attributes of an IDO are its primary key properties, which reflect the primary key properties of the IDO's base table in the application database. The system automatically detects primary key properties except for IDOs based on a SQL view. The order of the IDO's primary key properties should match the order used in the application database table. This order determines the default sort order for the IDO (which may be overridden in the application or elsewhere).

To add or edit primary keys for an IDO:

1 On the IDOs form, select the IDO you want.

2 Click **Check Out**.

3 Click **Advanced Attributes**.

4 In the Advanced IDO Attributes form, the Primary Keys list (on the right) indicates the primary keys detected by the system and their sort order. The IDO Properties list (on the left) lists the properties of the IDO. Use these techniques to work with primary keys:

- To add a property as a primary key, select the property you want in the IDO Properties list. Then click **Add**.
- To set the sort order for the primary keys, select a primary key in the Primary Keys list. Then click **Up** or **Down**.
- To remove a primary key, select the primary key in the Primary Keys list. Then click **Remove**.

5 Click **OK** to save your changes.

Editing IDOs

After you make changes in the application database (such as modifying tables or stored procedures), you must update (edit) any IDO that is associated with the changed tables or stored procedures.

To edit an IDO:

1. In the **IDOs** form, select the IDO you want to edit.
2. Click **Check Out**.
3. You can make many required changes directly on the **IDOs** form.

You might also need to open the IDO Tables form, the IDO Properties form, or the IDO Methods form and make any necessary adjustments there.

For example, you must update derived, subcollection, and unbound properties manually.

NOTE: You cannot change the IDO name.

Canceling or Undoing Changes to IDOs

After you check out an IDO and change it, you can cancel your changes before you check it back into the objects database.

To undo changes to the current IDO:

1. Open the **IDOs** form and select the desired IDO.
2. Select **Undo Check Out**.

You may have to refresh or close the form to see the change.

Deleting an IDO

Deleting an IDO removes all the methods and properties associated with it in the Objects database.

To delete an IDO:

1. On the **IDOs** form, select the IDO you want to delete.
2. Click **Check Out**.

   If an IDO has never been checked in, you can delete it without checking it in and then back out first.
3. Use the **Rules** tab to view or maintain the rules for what to do when an IDO related to this one is deleted.

   For more information, see **IDO Deletion Rules**.
4. Select Actions > Delete.

IDO Deletion Rules

For the deletion of IDOs, you can define zero or more rules. Rules are used identify:

- The IDO property or properties that are referenced by another IDO
- The referencing IDO
- The property or properties on the referencing IDO that correspond to the referenced properties

  For each rule, you can specify one of three actions to take when one or more records match the rule definition for deletion:

- **Restrict:** Aborts the deletion and displays a message indicating that the delete operation is not allowed

For example: "Cannot delete record. At least one sales order exists for this customer."

NOTE: For **Restrict** actions, you must specify translatable application message strings to be substituted in the delete operation.

- **Remove:** Allows the deletion but sets the referencing properties to NULL
- **Cascade:** Allows the deletion but first deletes any referencing records (subcollections)

You can optionally

- Specify the translatable message to display to the end user if a delete is prevented
- Supply a filter to further restrict the rows to which the delete policy applies

Exporting and Importing

Exporting and Importing IDOs

You can export an IDO and import it into a different Objects database.

To **export** an IDO:

1. On the IDO Export Wizard form, click **Browse** and specify a directory path and a filename for the export file to be created. The file type must be **.xml**.
2. In the IDOs to Export area, select the desired IDO. Then choose one of the following options:
   - To export all IDOs in the current project, choose **All IDOs in selected project**.
   - To export only some IDOs, choose **Selected** and then select the IDOs you want from the list.
   - Select **None** if you only want to export property classes or custom assemblies.
3. You can also choose to export selected property classes and custom assemblies.
4. When you have selected all the objects to export, click **OK**.

To **import** an IDO:

1. On the IDO Import Wizard form, click **Browse** and specify a directory path and a filename for the file to be imported. The file must have been created with the IDO Export Wizard, and the file type must be **.xml**.
2. The IDOs contained in the file you identified are displayed. In the IDOs To Import area, choose one of the following options:
   - To import all IDOs contained in the .xml file, select **All**.
   - To import only some IDOs, choose **Selected** and then select the IDOs you want.
   - Select **None** if only want to import property classes or custom assemblies.
3. You can also choose to import selected property classes and custom assemblies.
4. When you have selected all the objects to import, click **OK**.

Exporting and Importing Custom Assemblies

You can export an IDO custom assembly and import it into a different Objects database.

To **export** a custom assembly:

1. On the **IDO Export Wizard** form, click **Browse** and specify a directory path and a filename for the export file to be created. The file type must be **.xml**.
2. In the Custom Assemblies to Export area, select the desired IDO. Then choose one of the following options:
   - To export all custom assemblies in the current Objects database, select **All**.
   - To export only custom assemblies used by a specific IDO, select **Referenced by selected IDOs**. This option should be used in conjunction with exporting IDOs.
   - To export only some custom assemblies, select **Selected**. Then select the custom assemblies you want from the list.
   - Select **None** if you only want to export IDOs or property classes.
3. When you have selected all the objects to export, click **OK**.

To **import** a custom assembly:

1. On the IDO Import Wizard form, click **Browse** and specify a directory path and a filename for the file to be imported. The file must have been created with the IDO Export Wizard, and the file type must be **.xml**.
2. The custom assemblies contained in the file you identified are displayed. In the Custom Assemblies To Import area, choose one of the following options:
   - To import all custom assemblies contained in the .xml file, select **All**.
   - To import only some custom assemblies, choose **Selected** and then select the custom assemblies you want.
   - Select **None** if only want to import IDOs or property classes.
3. When you have selected all the objects to import, click **OK**.

Exporting and Importing Property Classes

You can export an IDO property class and import it into a different Objects database.

To **export** a property class:

1. On the **IDO Export Wizard** form, click **Browse** and specify a directory path and a filename for the export file to be created. The file type must be **.xml**.
2. In the Property Classes to Export area, select the desired IDO. Then choose one of the following options:
   - To export all property classes in the current Objects database, select **All**.
   - To export only property classes used by a specific IDO, select **Referenced by selected IDOs**. This option should be used in conjunction with exporting IDOs.
   - To export only some property classes, select **Selected**. Then select the property classes you want from the list.
   - Select **None** if you only want to export IDOs or custom assemblies.
3. When you have selected all the objects to export, click **OK**.

To **import** a property class:

1. On the **IDO Import Wizard** form, click **Browse** and specify a directory path and a filename for the file to be imported. The file must have been created with the IDO Export Wizard, and the file type must be **.xml**.
2. The property classes contained in the file you identified are displayed. In the Property Classes To Import area, choose one of the following options:
   - To import all property classes contained in the .xml file, select **All**.
   - To import only some property classes, choose **Selected** and then select the property classes you want.
   - Select **None** if only want to import IDOs or custom assemblies.
3. When you have selected all the objects to import, click **OK**.

Working with Tables

Understanding Tables Used by IDOs

Although there are many aspects of IDOs that do not correspond directly to tables in a SQL Server relational database, SQL Server tables provide the foundation on which any IDO is built.

- All tables used by IDOs must have certain required columns and triggers.
- Each IDO must have at least one base table.
- To include read-only information from associated tables in an IDO, use secondary tables.
- For most IDOs, primary key properties used by the base table in the application database are detected automatically. However, primary key properties for IDOs based on a SQL View must be specified manually. To identify the primary key properties and their sort order, set the IDO's advanced attributes.

About IDO Base Tables

Every IDO must have at least one base table. An IDO's base tables are used to store all updateable, persistent data associated with the IDO. When you create a new IDO, you are asked to provide the name of the IDO's primary base table. The primary base table is the central table to which any other table for the IDO is related in some way.

About IDO Secondary Tables

Sometimes you may want to include read-only information from associated tables in your IDO. Suppose, for instance, that you store a Unit of Measure code in your base table. The description associated with this code is stored in a separate Unit of Measure code table. You can publish the Unit of Measure description on your IDO by adding a secondary table to the IDO.

Columns from secondary tables can be published on the IDO as read-only properties. Secondary tables do not need to have the same primary key columns as the base tables, but you must specify the join criteria needed to include columns from secondary tables in your IDO.

Adding Base or Secondary Tables to IDOs

When you add an IDO, you must identify a primary base table.

If there are columns in other tables that have a relationship with the primary base table and if you want to expose these columns as properties on the IDO, you must also add the related table to the IDO as either a base table or a secondary table.

To add base or secondary tables to an IDO:

1. In the **IDOs** form, select the IDO to which you want to add tables.
2. Click **Check Out**.
3. Click **New Table**.
4. In the **New Table** form, enter the following information:
   - Table Name
   - Table Alias
   - Table Type
   - Join Type
5. If you are adding a secondary table, specify the join conditions.
6. Click **OK**.

WinStudio adds the table and join conditions to the IDO.

Including Data from a Different SQL Database into a Mongoose Application

To incorporate data from a non-Mongoose linked SQL Server database into a Mongoose-based application, you must specify information about the other database in the **IDO Linked Database** form and the **IDO Linked Tables** form.

NOTE: If you want to link to a non-Unicode database, use the **Process Defaults** form to set the process default for **Non Unicode Literal**. This helps ensure that the database indexes are scanned and accessed properly when performing queries. See the entry for **Non Unicode Literal** in the "Default Name" help topic.

To incorporate data from a non-Mongoose SQL database for use in a Mongoose-based application:

1. Open the **IDO Linked Database** form and specify these values:

   Link Database

   Specify the name that is to identify the database in Mongoose. This is a Mongoose internal designation only and need not be the same as the actual database name.

   Database Name

   Specify the SQL Server name of the database to which you want to link. If this database resides in the same location as the Mongoose databases, you can provide just the name of the database. If this database resides in a location other than the Mongoose databases, you must also provide the location of the database. Use this format: databaseServer.databaseName

   Optimistic Lock Column Name

Optionally, specify the name of a column that might be available for optimistic locking in the non-Mongoose database. If no specific column is designated on the IDO Linked Tables form, this value is used as the default optimistic lock setting.

2 Save your changes.

3 Open the **IDO Linked Tables** form to verify and adjust the column settings for the non-Mongoose table:

- Verify that the columns listed in the **Column Name** column match those of the non-Mongoose table.
- Optionally, rename the **View Column Names** as you want them to display in your Mongoose application.
- Verify that the primary keys for the non-Mongoose table are correct. Modify the choices for keys as necessary.
- Optionally, select the column to use for optimistic locking in the non-Mongoose database. Mongoose designates the “NODATE” literal string for the "RecordDate" value, and no optimistic locking is performed when these conditions exist:
  - No "RecordDate" column exists in the non-Mongoose table.
  - The column specified in the **Optimistic Lock Column Name** field of the **IDO Linked Databases** form does not exist in the non-Mongoose table.
  - No column is designated for optimistic locking on the **IDO Linked Tables** form.

4 To create the Mongoose view, click **Create View**.

Mongoose creates a view that includes the columns from the linked table, along with columns and values for:

- "RecordDate", used for optimistic locking
- "RowPointer", required to be a value that is unique for the entire table
- "AddMongooseFields", with a literal value of **1**, which is used internally

You can now use the linked database and tables in the same way that you use any database created within Mongoose. You can create IDOs and forms, and perform read-write operations on them like any other Mongoose database. However, you cannot use the non-Mongoose SQL database and tables for any event action where the workflow must be suspended. See the Guide to the Application Event System.

Editing or Removing IDO Tables

If tables in the application database change, you must update the table definitions in any IDO associated with those tables. You can edit or remove tables from an IDO, but you cannot remove the primary base table.

To edit or remove a table from an IDO:

1 On the **IDOs** form, select the IDO you want.

2 Click **Check Out**.

3 Click the **Tables** button to display the IDO Tables form. On this form:

- To edit a table (including the join conditions), select the table and click **Edit Table** to display the Edit Table form.
- To delete a table, select the table and then select **Actions > Delete** or click the Delete toolbar button.

Specifying Join Conditions for Secondary Tables

When you work with secondary tables, you can specify join conditions by selecting parameters from controls in the **Edit Table** form. For more complex join conditions, you can enter the join conditions manually.

To specify join conditions by selecting parameters:

1. In the IDO Tables form, click **Add** or **Edit** to add or edit a secondary table. The Edit (or New) Table form is displayed.
2. In the first Join Conditions field, select the column in the current table being joined to.
3. In the second Join Conditions field, select the table being joined.
4. In the third Join Conditions field, select the column being joined.
5. Click **Add** to display the conditions in the large edit box.
6. Click **OK**.

To manually enter join conditions:

1. In the IDO Tables form, click **Add** or **Edit** to add or edit a secondary table. The Edit (or New) Table form is displayed.
2. Enter your join conditions as a SQL SELECT statement in the large edit box. If necessary, you can use the three Join Conditions boxes to select columns and tables and insert them as parameters in your SQL statement. The parameters are inserted as AND statements.
3. Click **OK**.

Tip: Using Multiple Base Tables

Most IDOs should have only a single base table. Sometimes, however, you may find it useful to implement an IDO with multiple base tables. Normally, multiple base tables are required only when you are working with an existing database schema that was designed for other purposes.

Nonprimary base tables function as extensions to the primary base table. They must have the same key columns as the primary base table. The nonprimary base table may store either **optional** data (meaning that a corresponding record may or may not exist for each record in the primary base table) or **required** data (meaning that a corresponding record must exist for each record in the primary base table). If a nonprimary base table stores optional data, then it must be joined using a left outer join.

It is the responsibility of the Insert trigger on the primary base table to insert records into each required base table.

It is the responsibility of the Delete trigger on the primary base table to delete records in all related base tables, whether they are optional or required.

Working with Methods

Understanding IDO Methods

IDOs have two kinds of methods, standard methods and custom methods.

Standard Methods

All IDOs implement the methods **LoadCollection**, **UpdateCollection**, **GetPropertyInfo**, and **Invoke**.

- **LoadCollection** retrieves a collection of rows from the database.
- **UpdateCollection** takes a set of rows marked for insert, update, or delete, and executes the appropriate SQL code against the database.
- **GetPropertyInfo** returns detailed information about the properties supported by the IDO.
- **Invoke** allows you to execute a custom method.

Custom Methods

Custom methods are defined by the developer. You can define custom methods that are implemented in Transact-SQL. Transact-SQL is the preferred programming language because it is easier to use for most of these tasks and because the IDO forms provide useful facilities for defining methods based on these procedures.

There are two kinds of methods based on Transact-SQL stored procedures: methods based on stored procedures without a result set and methods based on stored procedures with a result set.

A stored procedure that does not return a result set may have input and output parameters, but it does not select data to be returned to the caller.

A method that returns a result set (a list of values) to the caller may be used to populate collections. A typical use is to populate drop-down list boxes in forms. You can also bind forms to these returned sets.

About IDO Extension Classes

An IDO extension class is a .NET class that allows developers to extend the functionality of an existing IDO by adding methods and event handlers. IDO extension classes are compiled into a .NET class library assembly and stored in the IDO metadata database. The IDO runtime loads these assemblies on demand and calls methods and event handlers in the extension classes in response to IDO requests.

An extension class is short-lived; it is created at the start of a request and disposed of immediately when the response is completed. Therefore, no state should be stored in an extension class.

Any public class in an IDO extension class assembly can be identified as the extension class for an IDO in the IDO metadata database. IDO extension class assemblies are created from VisualStudio 2005 class library projects. The projects can use any of the .NET languages.

Transactions and IDO Methods

IDO methods can be run within the context of a transaction. To reduce transactional overhead and blocking, you can set a method to run without a transaction.

Methods that do not update data in the database can likely be run without a transaction. A method that runs a long time in a transaction, potentially blocking other transactions, may perform better if it is run without a transaction. You can control the size of individual transactions by starting them and committing or rolling them back within a stored procedure.

To set a method to run within a transaction, select the **Transactional** check box on the **IDO Methods** form. At run time, the method starts a transaction before the stored procedure is called.

To set a method to run without a transaction, clear the **Transactional** check box. The method does not start a transaction before the stored procedure is called.

Adding a Method to an IDO

A method is based on a stored procedure in the application database or an extension class method.

To add a method to an IDO:

1. Open the **IDOs** form.
2. Select an IDO and click **Check Out**.
3. Click **New Method**.
4. From the **Method Type** drop-down list in the **New Methods** form, select the type of procedure being called by this method. You can specify a method that either does or does not return a result set.
5. If applicable, select the **Stored Procedure** from the list of stored procedures in the application database.
6. In the **Method Name** field, enter the name of the new method. This name must be unique to this IDO.
7. Select **Transactional** if the method should be run as a transaction.
8. Click **OK**.

After you refresh, the new method is displayed in the **IDOs** form's **Methods** tab and is added to the IDO.

Editing Methods

After you modify a stored procedure in the application database, any method that is associated with the stored procedure needs to be updated to reflect the changes.

To edit a method:

1. Open the **IDOs** form and select the IDO you want.
2. Click **Check Out**.
3. Click **Methods**.

4 In the IDO Methods form, select the method you want to update.

5 Enter and save your changes.

Deleting a Method or Property from an IDO

You can delete a method or property associated with an IDO.

To delete a method from an IDO:

1 In the **IDOs** forms select the IDO you want.

2 Click **Check Out**.

3 Click **Methods...**

4 Select the method to delete.

5 Select **Actions>Delete** or click the Delete button on the toolbar.

To delete a property from an IDO:

1 In the **IDOs** form, select the IDO you want.

2 Click **Check Out**.

3 Click **Properties...**

4 Select the property you want to delete.

5 Select **Actions>Delete** or click the Delete button on the toolbar.

Working with Properties

Understanding Properties

Bound Property

Bound properties are persistent properties whose values are stored in an application database table. This is the most common type of property.

Derived Property

Derived properties are properties whose values are derived from SQL expressions. Use derived properties to calculate values, to execute subqueries, or to call SQL functions.

Unbound Property

Unbound properties are properties whose values are not stored in a database table and consequently are not persistent. Use this type of property to pass temporary values from a form to an IDO. This data can be used by a custom insert, update, or delete method.

Subcollection Property

A subcollection property is a property that specifies a child IDO that is filtered from a parent IDO. A subcollection is the child IDO whose returned collection is associated with, and dependent on, the

objects returned in the primary collection belonging to the parent IDO. Subcollections are the principal mechanisms for defining hierarchical or parent-child data relationships. Use subcollections to implement one-to-many relationships between IDOs.

The typical implementation of order lines in a business application is a good example of a subcollection. Each order returns a collection of order lines. The system would define the order lines as a subcollection of the order IDO.

Note that order lines are dependent on orders; that is, order lines cannot exist independently of their parent orders. However, this is not a requirement of subcollections in general. For instance, you may define a collection of customers as a subcollection of the collection of account managers. Each account manager has a set of customers that he or she services. Customers can exist totally independently of their account managers. It should be possible, for instance, to move a set of customers to a new account manager and delete their old account manager from the system.

Subcollections can also be used to implement recursive data structures. A good example of this is a typical implementation of a product structure or bill of materials. A product structure record typically includes a reference to a set of child entities that are themselves product structure records. In this case, you could define a product-structures IDO with a subcollection of product structures.

To define a subcollection, you must first define the collection class that characterizes the child IDO. Then you must establish the relationship between the new child IDO and the parent IDO by creating a subcollection property on the parent. In the case of orders and order lines, you would define an orders collection and an order-lines collection. Then you would define the relationship between the two by defining a subcollection property on the orders IDO.

Adding a Property to an IDO

To add a property to an IDO:

1. Open the **IDOs** form.
2. Select an IDO and click **Check Out**.
3. Click **New Property**.
4. In the **New Property** form, select a type of binding for the new property, either Bound, Derived, Unbound, or Subcollection.
5. Click **Next**.
6. Set the attributes for the type of property you selected.
7. Click **Finish**.

Editing an IDO (Table) Property

You can edit the IDO property for any table in the IDO. For example, you should edit bound properties after the table column bound to the property is modified in the application database.

To edit an IDO property:

1. In the **IDOs** form, select the IDO you want.
2. Click **Check Out**.

3 Click **Properties**. Expand the **Properties** folder and select the property you want to update.

4 In the **IDO Properties** form, select the property you want to update and enter your changes.

Deleting a Method or Property from an IDO

You can delete a method or property associated with an IDO.

To delete a method from an IDO:

1 In the **IDOs** forms select the IDO you want.

2 Click **Check Out**.

3 Click **Methods...**

4 Select the method to delete.

5 Select **Actions>Delete** or click the Delete button on the toolbar.

To delete a property from an IDO:

1 In the **IDOs** form, select the IDO you want.

2 Click **Check Out**.

3 Click **Properties...**

4 Select the property you want to delete.

5 Select **Actions>Delete** or click the Delete button on the toolbar.

Exporting and Importing Property Classes

You can export an IDO property class and import it into a different Objects database.

To **export** a property class:

1 On the **IDO Export Wizard** form, click **Browse** and specify a directory path and a filename for the export file to be created. The file type must be **.xml**.

2 In the Property Classes to Export area, select the desired IDO. Then choose one of the following options:

- To export all property classes in the current Objects database, select **All**.
- To export only property classes used by a specific IDO, select **Referenced by selected IDOs**. This option should be used in conjunction with exporting IDOs.
- To export only some property classes, select **Selected**. Then select the property classes you want from the list.
- Select **None** if you only want to export IDOs or custom assemblies.

3 When you have selected all the objects to export, click **OK**.

To **import** a property class:

1. On the **IDO Import Wizard** form, click **Browse** and specify a directory path and a filename for the file to be imported. The file must have been created with the IDO Export Wizard, and the file type must be **.xml**.
2. The property classes contained in the file you identified are displayed. In the Property Classes To Import area, choose one of the following options:
   - To import all property classes contained in the .xml file, select **All**.
   - To import only some property classes, choose **Selected** and then select the property classes you want.
   - Select **None** if only want to import IDOs or custom assemblies.
3. When you have selected all the objects to import, click **OK**.

Creating an Inline List

NOTE: This topic applies to the creation of inline lists to use in conjunction with IDO properties and property classes. For the process to create an inline list at the component level in Design Mode, see the topic "**Error! Hyperlink reference not valid.**."

An inline list is a set of "hard-wired" values for a property that can be applied to a component. Such a list is typically used when the number of values to appear in a combo box or other display is limited and the values will be constant and unchanging.

To create an inline list for an IDO property or property class:

1. On the **IDO Properties** form or the **Property Classes** form, click the ellipses button (...) to the right of the **Inline List** field.
2. In the **Edit Inline List** form, determine the size of your inline list by adding rows and/or columns.

   You can add any number of rows or columns, but for inline lists, it is usually a good idea to keep the number small. You should decide before creating the list exactly how many rows and how many columns your list needs to have.
3. In the **Column For Value** field, specify by column number which column is to provide the values to any components that use this property or property class.
4. In the **Display Columns** field, specify by column number which columns are to have their values displayed when the list is used.

   If you want to display multiple columns, separate the column numbers with commas.

Be aware that, although you can use translatable strings in these lists, the IDO Runtime Service cannot access the string IDs. You can, however, resolve the string IDs at the user interface level.

Example: Suppose you want to create an inline list with:

- A set of four severity levels: Low, Medium, High, and Severe
- A translatable string value for each level: sLow, sMedium, sHigh, sSevere
- A numerical value for each severity level, so that the list items can be presented in the correct order, regardless of alphabetization or translation concerns

In this case, you would require two columns, one for the translatable strings and the other for the numeric values. You would require four rows, one for each severity level.

Then, in the **Column For Value** field, you would want to specify the numeric column as the value for the component using this property to use in creating the list in the correct order.

And finally, in the **Display Columns** field, you would want to specify the column containing the strings as the values to display when the list is actually presented in the UI.

When you click **OK** in the **Edit Inline List** form, then, WinStudio creates this metadata in the **Inline List** field:

ENTRIES(sLow\0,sMedium\1,sHigh\2,sSevere\3) DISPLAY(1) VALUE(2)

Creating an IDO Property Validator

NOTE: This topic applies to the creation of validators to use in conjunction with IDO properties and property classes. For the process to create a validator at the component level in Design Mode, see the topic **Error! Hyperlink reference not valid.**.

Validators are global objects that can be used to validate input or other actions taken by a user.

To create and assign a validator as part of an IDO property or property class:

1. On the **IDO Properties** form or the **Property Classes** form, click the ellipses button (...) to the right of the **Validators** field.
2. On the main page of the form, click **Add**.
3. On the **Select Validator Type** page, select the type of validator you want to add.
4. Click **Next**.
5. The page you proceed to depends on the type you selected. If you selected Inline List:
   a. Click **Add**.
   b. In the **Property Name** field, specify the property you want to set a value for if the validation is successful.
   c. In the **Inline List Column** field, specify the index of the column that is to contain the value of the component bound to the property.
   d. Optionally, in the **Error Message** field, specify an application error message to display if validation is not successful.
   e. Click **OK** and go on to Step 8.
6. If you selected IDO Collection:
   a. In the **IDO Name** field, specify the IDO to use for validation.
   b. In the **Property Name** field, specify a property that you want to validate from the IDO collection named in the **IDO Name** field.
   c. Optionally, in the **Filter** field, specify the IDO filter you want to use during validation.
   d. Click **Add**.

e In the **Target Property** field, specify the property to be updated from the IDO collection that is being saved and validated.

f In the **Source Property** field, specify the property from the IDO being used for validation that contains data to be copied into the target property.

g Optionally, in the **Error Message** field, specify an application error message to display if validation is not successful.

h Click **OK** and go on to Step 8.

7 If you selected IDO Method:

a In the **IDO Name** field, specify the IDO that contains the method you want to validate.

b In the **Method Name** field, specify the method to be evaluated by this validator.

c Optionally, click **Add**. Then:

- In the **Type** field, specify whether the designated parameter is to be used for input only, both input and output, or a message.
- In the **Source** field, specify whether the parameter is to treat the value as a property value or treat it as a literal value.
- In the **Value** field, specify the value for the parameter that is to be passed to the method.

d Optionally, in the **Error Message** field, specify an application error message to display if validation is not successful.

e Click **OK** and go on to Step 8.

8 If you are finished adding validators, to return the entire validator specification to the **Validators** field, click **OK**.

Form Control

Understanding Form Control

Form Control is a version-control tool for objects being developed for the presentation layer (client tier) of an application in WinStudio. WinStudio is a form presentation and editing engine used to create and modify forms and global objects. These objects are stored in the forms and templates databases that are configured for WinStudio.

Form Control is used to access the objects stored in these forms and templates databases. Form Control tracks versions of objects and supports access to a repository of versions and an archive of deleted objects. Although implementing Form Control is optional, this tool is particularly useful when several developers are working on an application, as it allows one developer to lock a form or global object while working on it, so that other developers cannot work on the same object at the same time.

Use Form Control to check out obpjects from the "master" databases (which contain the current versions of objects) to "run-time" databases (which temporarily store the objects) for editing. After editing the objects in WinStudio, you then use Form Control to check the edited objects back in to the master databases. If you use a source control system, Form Control also checks objects in to source control when you check objects in to the master databases.

You can also delete objects from the master and run-time databases and can copy them to archive databases. These previous versions of objects are then available through source control.

NOTE: You can use Form Control with login IDs having Site Developer editing permissions. When you log in to Form Control, make sure you select **Site** in the **Scope** field.

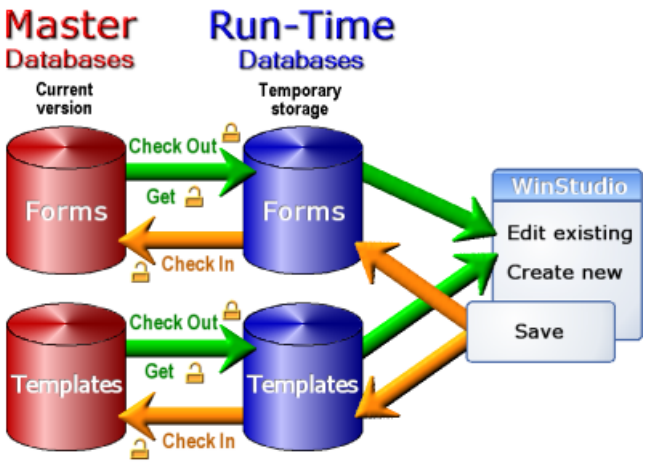
Before You Begin

Before you can use Form Control, you must have at least one system configuration set up using Configuration Manager. Use Configuration Manager to create a configuration, including a tools login that is used to open Form Control and to track check-ins and check-outs. The tools login must have Vendor Developer editing permissions in WinStudio.

For more information about setting up configurations, see the help for Configuration Manager.

Basic Functionality of Form Control

This diagram illustrates the basic functionality of Form Control:



Use Form Control to check out (from the master databases) any forms or global objects that you want to edit. Form Control places copies of the objects in the run-time database while you have them checked out.

After you check out an object, use WinStudio in Design Mode to make the desired changes. You can also use WinStudio to create new objects. In either case, saving changes in WinStudio stores the changes in the run-time database copy.

When you are finished working with the object in WinStudio, use Form Control to check the object back in to the master database. You can check in at one time both objects that you checked out and new objects you have created.

While using Form Control, keep in mind that:

- Form Control works only with the vendor and site default versions of a form or global object. Only developers with vendor or site developer editing permissions can create vendor or site default versions.
- Checking out an object locks it for your use and prevents other developers from checking it out or saving other changes to it.

  NOTE: Other developers can *view* and work with the form or object in Design Mode, but they cannot save their changes until you check it back in.
- Checking in objects that you checked out removes the lock on the object.
- If you create and save a new object, WinStudio saves the object in the appropriate run-time database. It cannot be overwritten, because the object does not yet exist in the master database (until you check it in).
- Checking in new objects adds them to the master databases and allows Form Control to track them.
- If you make and save changes to an object that is not checked out (to you or anyone else), your changes will be overwritten the next time anyone checks that object out. To help prevent this from happening, you can set a user preference for WinStudio to warn developers about this type of situation.
- If Source Control is enabled for a configuration, Form Control also checks objects in to source control when you check objects in to the master databases.

Tips

- If you cannot check out an object because it is locked, Form Control displays the user ID of the developer who has the object checked out.
- When you check out a form, you can see a list of all the associated global objects and can check out all or some of those objects at the same time.

  For more information, see Checking Out Forms.
- When you check out a form, you can also check out the form template associated with it.

  For more information, see Checking Out Forms.
- You can check out global objects independently, regardless of the forms they are associated with.

  For more information, see Checking Out Global Objects.
- You can display a report of all the objects you have checked out.

  For more information, see Displaying the Current Status of Forms and Global Objects.
- If you want to examine or test objects, you can get a copy of the current version of the object without checking it out. In this way, many developers can get the same object, as long as no one has the object checked out. If changes are made, they are not permanent because they cannot be checked in.
- You can archive a form or global object. Archiving deletes the object from both the master and run-time databases and moves it to the corresponding archive database.

  For more information, see Archiving Forms or Archiving Global Objects.

Form Control Tasks

Form Control is essentially a file and version-control manager. You can use it to control who can and cannot work on which forms or global objects in a WinStudio development environment. As such, it is limited to the following basic tasks:

- Checking objects out
- Checking objects in
- Getting objects
- Unlocking objects
- Archiving objects
- Restoring objects
- Displaying status of objects

About Checking Out Objects

Checking out a form or global object allows you make changes to the object in WinStudio, without having to worry about others overwriting your changes. When you check out an object, Form Control locks the object so that no one else can make permanent changes to it until you check it back in or unlock it.

You can check out an object only if no one else already has it checked out.

You can check out:

- Forms, with or without their associated templates and/or global objects

  For the procedure, see Checking Out Forms.

- Global objects, regardless of whether or not they are associated with forms

  For the procedure, see Checking Out Global Objects.

During a checkout procedure, Form Control:

- Verifies that no one else has the object checked out

  If an object is checked out already, the system displays a message identifying who has it checked out. You cannot check out the object until the other developer checks it in.

- Warns if device types exist for a form but are not synchronized between databases. For more information, see Utilities Tab.
- Copies the object from the master database to the appropriate run-time database and locks it in the master database

  That is, the **LockBy** column contained in the forms or templates database is set to your user ID, indicating that you have the item checked out. WinStudio prevents other developers from checking out or saving changes to the vendor default version of any objects that you have checked out.

About Checking In Objects

Checking in a form or global object replaces the original version in the master database with the edited version from the run-time database. This action effectively makes the changed object the official current version.

You can check in an object only if one of these situations is true:

- You have it checked out.
- It is new and does not exist in the master database.

You can check in:

- Forms, with or without their associated templates and/or global objects.

  For the procedure, see Checking In Forms.

  NOTE: To check in associated templates and/or global objects, you must have those objects checked out as well.

- Global objects, regardless of whether they are associated with forms or not.

  For the procedure, see Checking In Global Objects.

During a check-in operation, Form Control:

- Verifies that any objects you are trying to check in were checked out to you, or that the objects are new and do not exist in the master databases.
- Copies the objects from the run-time databases to the appropriate master databases.
- If your system is integrated with a source control system, checks the objects into source control.

  Following check-in, Form Control displays a dialog box that indicates any generated files that were not checked in or added to source control.

- Unlocks the objects in both the master and run-time environments.

  That is, the **LockBy** column contained in the form and templates databases is set to an empty string, indicating that no one has the item checked out.

About Getting Objects

Getting a global object allows you to retrieve a copy of the object without putting a lock on it or formally checking it out. Getting an object that no one has checked out enables several people to work with it at once, but no changes can be made permanent without actually checking out the object. So, typically, you would get an object (as opposed to checking it out) only when you want to make changes of an experimental nature without worrying about the changes becoming permanent.

NOTE: Both getting and checking out an object overwrite any existing copies in the run-time database with the version currently in the master database. This means that you cannot make changes made to a "get" version permanent, as you cannot check in a "get" version, and any changes you make are overwritten when you check it out.

You can get:

- Forms, with or without their associated templates and/or global objects

  For the procedure, see Getting Forms.

- Global objects, regardless of whether or not they are associated with forms

  For the procedure, see Getting Global Objects.

During a get operation, Form Control:

- Verifies that no one has the object checked out

  If an object is checked out by someone other than you, you cannot get it. (The **Execute Action** button is disabled in this case.)

  CAUTION: If the object is already checked out to you, you can get the object, but doing so overwrites whatever changes you have made to the object since checking it out. In effect, this reverts the object to the state it was in when you checked it out.

- Copies the object from the master database to the appropriate run-time database

About Unlocking Objects

Unlocking an object effectively reverts it to the state it had before you checked it out. If you are using a source control system, this also undoes the checkout for source control. This is typically most useful when you have made changes, and then you decide you do not want to keep them.

You can unlock only objects that you have previously checked out. You cannot unlock objects that others have checked out, nor can they unlock yours.

You can unlock:

- Forms, with or without their associated templates and/or global objects

  For the procedure, see Unlocking Forms.

- Global objects, regardless of whether or not they are associated with forms

  For the procedure, see Unlocking Global Objects.

During an unlocking operation, Form Control:

- Verifies that any objects you are trying to check in were checked out to you
- Unlocks the objects in both the master and run-time environments

That is, the **LockBy** column contained in the form and templates databases is set to an empty string, indicating that no one has the item checked out.

About Archiving Objects

Archiving a form or object removes it from the active development environment and places it in a special archive database. Generally, you archive objects only when you are no longer using them but want to keep copies for reference or possible later restoration.

You can archive objects only if they are not checked out by anyone.

You can archive:

- Forms, with or without their associated templates and/or global objects

  For the procedure, see Archiving Forms.

- Global objects, regardless of whether or not they are associated with forms

  For the procedure, see Archiving Global Objects.

During an archive operation, Form Control:

- Moves the object to the appropriate archive form or templates database
- Deletes the object from the master and run-time databases

About Restoring Objects

Restoring a form or object retrieves it from the archive database and places it back in the appropriate master database.

You can restore only objects that have been previously archived.

You can restore:

- Forms, with or without their associated templates and/or global objects

  For the procedure, see Restoring Forms.

- Global objects, regardless of whether or not they are associated with forms

  For the procedure, see Restoring Global Objects.

During a restore operation, Form Control:

- Retrieves the object from the appropriate archive form or form templates database
- Copies the object to the appropriate master database

Event System

Events Overviews and Processes

Additional information about the application event system can be found in the *Guide to the Application Event System*, available on our Support site.

Creating Events

You can create your own custom events to use as part of the event system.

To create an event:

1. Open the **Events** form.
2. Verify that Filter-in-Place is off, and then select **Actions** > **New**.
3. In the **Event Name** field, enter the name by which the event will be identified in the system.
4. (Optional) In the **Description** field, enter a functional description that will help to identify the event's intended use.
5. Save the form.

Creating Event Handlers

In its most basic form, an event handler consists of the following:

- An association with a particular event
- A handler sequence number
- One or more actions to perform when the event is triggered

To create an event handler:

1. Open the **Event Handlers** form.
2. Press F3.
3. Select **Actions** > **New**.
4. In the **Event Name** field, do one of the following:
    - Select the event for which you want to create a handler. This sets up the event handler for an already-defined event.

- Enter the name for an event which has not been previously defined.

5 Enter a description for the handler. This description is helpful when you have multiple handlers with the same name, because it allows you to find the appropriate handler in a list.

6 (Optional) To control the handler sequence with respect to other developers' handlers, use the **Keep With** and **Chronology** fields. For more information, see **Keep With** and **Chronology**.

7 Save the new event handler.

8 Use the **Event Actions** button to open the **Event Actions** form and create the actions to be performed when this event handler is executed.

9 After closing the **Event Actions** form, set the other options on the **Event Handlers** form as desired. For information about the other options, see the online help for each option.

10 Save the event handler.

Note: Events and event handlers are defined for an application database, not a particular site. To allow an event handler to run only at certain sites on a database, specify the sites in the **Applies To Sites** field on the **Event Handlers** form.

For more information, see "About Event Handlers" in the *Guide to the Application Event System*.

To see a graphical (diagrammatic) representation of the event handler and its actions, click the **Diagram** button, which opens the **Event Handler Diagram** form. This form also allows you to edit and even add event actions to the event handler flow.

Using the Event Handler Diagram Form

NOTE: Event diagramming uses the *Nevron Diagram for .NET* tool. For more information, including complete documentation, please visit www.nevron.com.

You can use the **Event Handler Diagram** form to:

- View a graphical (diagrammatic) representation of an event handler and its actions.
- Access the **Event Actions** form to modify event actions.
- Add event actions to the flow.
- Delete event actions from the flow.
- Edit the final diagram before copying, printing, or saving.
- Copy the diagram to the system clipboard.
- Print the diagram.
- Save the diagram as a graphics file.

Viewing an Event Handler Flow

There are two ways to call up and view an event handler flow:

- In the **Event Handlers** form, select the handler for which you want to view the flow and then click the **Diagram** button.
- Open the **Event Handler Diagram** form, then select an event handler (**Event Handler Name**) and **Sequence** number.

Accessing the Event Actions Form to Modify Event Actions

To access the **Event Actions** form for a selected event action in the flow:

1. In the diagram display, double-click the node for the event action you want to edit.
2. When the **Event Actions** form opens, edit the action as desired.
3. Save your changes and close the form.
4. When the system asks whether you want to update the event handler diagram with the latest changes, respond **Yes**.

Adding Event Actions

To add an event action to an event handler flow:

1. Click the Event Actions menu bar on the left edge of the diagram display area.
2. Drag and drop the new event action into the diagram display area.
3. When the **Event Actions** form opens, use it to define the new action.
4. When finished, save the new action and close the form.

Deleting Actions

To delete an event action from an event handler flow:

1. In the diagram display, select the node for the event action you want to delete.
2. Right-click the node.
3. From the context menu, select **Delete Event Action**.

Editing the Diagram Display

To edit the diagram display:

1. Select the node that you want to move.
2. Click-and-drag the node to the desired location.

Any changes you make to the diagram are not persistent; that is, they only last as long as you make no more changes to the event handler flow. Therefore, if you want to make manual adjustments to

the layout of the diagram before printing it, for example, these manual adjustments should be the very last thing you do.

Copying the Diagram to the System Clipboard

To copy a diagram to the system clipboard:

1. Right-click anywhere inside the diagram display area.
2. From the context menu, select **Copy to Clipboard**.

Once the diagram is copied to the system clipboard, you can paste it into another program, such as MS Word, PowerPoint, or a graphics program.

Printing a Diagram

To print a diagram:

1. Right-click anywhere inside the diagram display area.
2. From the context menu, select **Print**.

The system prints the diagram on whatever printer is designated as the user's default printer.

Saving the Diagram

To save a diagram as a graphics file:

1. Right-click anywhere inside the diagram display area.
2. From the context menu, select **Save To Image**.
3. In the **Save As** dialog box, provide a name for the image file and then click **Save**.

   CAUTION: Although other file types might appear to be available, you can only save the diagram as a .BMP file type. Do not select any other file type, or errors might occur.

Creating Event Actions

To be useful, each event handler must have at least one event action associated with it.

An event action is defined as a unit of work to be done when the event handler executes.

To create an event action:

1. Open the **Event Handlers** form.
2. Create a new event handler; or, in the grid view, select the event handler for which you want to create the action.
3. Click the **Event Actions** button.

4 If you are creating an action for an event handler that already has actions defined, in the grid, create a new row.

5 In the **Event Actions** form, **Action Type** field, select the type of action to be performed.

For a list of action types and what they do, see Action Type.

6 Enter a description for the action. This description is helpful when you have multiple actions with the same name, because it allows you to find the appropriate action in a list.

7 Click the **Edit Parameters** button.

Based on the action type you selected, the system opens the appropriate event action parameter form, which you can use to construct the parameters for the action type. For more information, see Setting Event Action Parameters.

NOTE: If you are familiar with the parameters, functions, and syntax for the action parameters, you can click the **Show Details** button to display tabs where you can manually enter the parameter information in the text field. However, unless you are very confident in your ability to write this data from scratch or you are pasting in data from a reliable source, and to help ensure that you use only valid parameters, functions, and syntax, we recommend that you use the event action parameter forms, which have been designed specifically for this purpose.

For more information about the event action parameter forms, see the online Help for each form. For additional reference information about action types and parameters, see "Event Action Types" in the *Guide to the Application Event System*.

8 When you are finished setting up parameters in the appropriate event action parameter form, click **OK**.

Closing an event action parameter form automatically returns the parameters you set up there to the **Event Actions** form, with the correct syntax. Even so, we recommend that you verify the syntax is error-free by clicking the **Check Syntax** button before proceeding.

9 If the action involves a variable to be used in event messages and you want to restrict how the variable's value is treated, set those restrictions on the **Variable Access** tab.

For more information about setting variable access, see Event Actions - Variable Access Tab.

10 Save your changes and close the form.

For more information, see "About Event Actions" in the *Guide to the Application Event System*.

Setting Event Action Parameters

There are two basic ways you can set parameters for an event action:

- Use the event action parameter forms associated with each action type.

  Access these forms by first selecting an **Action Type** and then clicking **Edit Parameters** on the **Event Actions** form.

- Enter the parameters directly in the text edit field on the **Event Actions** form.

You can also begin with the event action parameter forms and then manually edit the output in the **Event Actions** form.

NOTE: If you are familiar with the parameters, functions, and syntax for the action parameters, you *can* manually enter the parameter information in the text field. However, unless you are very confident in your ability to write this data from scratch or you are pasting in data from a reliable source, and to help ensure that you use only valid parameters, functions, and syntax, we recommend that you use the event action parameter forms, which have been designed specifically for this purpose.

Tips and Guidelines for Using the Event Action Parameter Forms Effectively

While the event action parameter forms make it easier to set event action parameters than creating them manually, you must still be somewhat familiar with the parameters, functions, and syntax available for each action type. Probably, the best way to do this is to open the form associated with each action type and access the online Help for that form and its fields. There is also extensive reference documentation provided in the *Guide to the Application Event System*.

Each event action parameter form includes only those parameters and functions that will work for the selected action type. So, for example, if you are creating an action to notify recipients of something, only the parameters you might need to create that notification are available from the **Event Action Notify** form.

Most options on an event action parameter form include both a field and a button. The field might be:

- A text edit field into which you can directly enter the value for that option.
- A drop-down list from which you can select the value you want.
- A combo box that allows you to either select a value from a drop-down list or enter the value manually.

The associated button typically opens either of the following:

- The **Event Action Expression Editor**, which is a generic form used to create desired values using expressions. For more information, see Using Expressions in Event Action Parameters.
- Another auxiliary event action parameter form specifically designed to help with the creation of an appropriate value for that option.

For example, if you click the **Condition** button that appears on many event action parameter forms, the system opens the **Event Action Parameter Condition** form, which is designed specifically to make it easier to create and format an appropriate condition statement for the event system to use.

When you click **OK** in the event action parameter form, the system returns the values you specified, formatted using the correct syntax.

We recommend that you verify the syntax is error-free by clicking the **Check Syntax** button before proceeding.

Example

As part of an event that notifies a manager when a customer's credit limit has been changed, suppose you want to prompt a Credit Manager for approval if the new credit limit is $500,000 or less.

If the new credit limit is more than $500,000, then the Credit Supervisor must approve the change. You could use a Branch action type to determine who gets the prompt message.

To handle this situation, you would create an event handler that runs every time a customer's credit is changed. After using the **Event Handlers** form to create the handler, you would then click **Event Actions** to open the **Event Actions** form. There, for one of the actions, you would assign an action sequence number and select **Branch** in the **Action Type** field.

At this point, you would click **Edit Parameters** to open the **Event Action Branch** form. That form has two fields and one button (besides the **OK** and **Cancel** buttons). Since you want to use the auxiliary form to set the condition, you would click the **Condition** button.

In this example, the condition must be set so that if the value of the CreditLimit property is less than or equal to than 500,000, then the event handler proceeds to the next action step. Otherwise, if the CreditLimit property is greater than 500,000, then the action flow branches to a different action step.

When the **Event Action Parameter Condition** form opens, you decide to use the **Expression** buttons to help set up the condition. Clicking the **Expression 1** button opens the **Event Action Expression Editor**, which you will use to set up the first part of the condition. You want to specify the value of the **CreditLimit** property, so you select **PROPERTY** (or **P**) from the **Select a function** drop-down list.

After you select the **PROPERTY** function, the **Event Action Expression Editor** adds a **Parameter 1** field. In this field, because you know the name of the property for which you want to return the value, you would simply enter **CreditLimit** and then click **OK** to return to the **Event Action Parameter Condition** form.

In the **Operator** drop-down list, you would then select the "greater than" (>) symbol.

Finally, in the **Expression 2** field, you would enter **500000**. In this case, you do not need to use the **Event Action Expression Editor**, because you know the value you want to use, and you want to "hard-code" it to that value.

When you click **OK**, the system returns you to the **Event Action Branch** form and populates the **Condition** field. In the **Destination** field, then, you would either:

- Select the number of the action sequence step you want the handler to go to if this condition tests TRUE (if the target action sequence step already exists).
- For an action sequence step that does not yet exist, enter the number of the step you plan to create for the target later.

When you click **OK**, the system returns to the **Event Actions** form, with the correct parameter text.

At this point, we recommend that you click **Check Syntax** to verify that the parameter syntax is all correct. If there is an error, you will see a rather long and involved error message. You can use this error message to determine where and what the error is likely to be by concentrating on the line on which the error appears and the "Preceding context". In many cases, such as this one, you must have the property name in quotation marks, or the system returns an error. (In fact, missing quotation marks are the most common cause of syntax errors.)

If this happens, try to correct the error manually and then click **Check Syntax** again. Keep doing this until you have eliminated any errors.

CAUTION: If you do not correct any syntax errors before clicking **Edit Parameters** again, you will lose all parameter text and need to start over.

Using Expressions in Event Action Parameters

Many,but not all, event action parameters allow you to use expressions, rather than literal values, to specify the values of parameters. You typically do this when you want to allow for variable or dynamic values to be used for these values.

For example, you might have a case where you want to specify a group of recipients to receive various notifications and prompt messages, and that group membership changes often. In such a case, you can create a global constant value for the group and then use that global constant whenever you want a message sent to that group. Then, when the group membership changes, you can change it in one place (the global constant) and all event handlers that use that global constant automatically pick up the change.

Syntax for Expressions

The system recognizes a literal string value by the use of quotation marks. For numbers, dates, or Boolean values (TRUE/FALSE), the system does not require or allow quotation marks.

Consider the following examples:

- CONDITION("CreditLimit" < "500000")

  In this example, the system recognizes both sides of the comparison as literal string values and treats them accordingly when executing the event action. The values are compared alphabetically as strings. Therefore, this condition results in a false result, because "C" sorts higher than "5" in the Unicode collation.

- CONDITION(CreditLimit < 500000)

  In this second example, the system returns a syntax error, because it does not recognize **CreditLimit** as a valid function.

- CONDITION(P("CreditLimit") > "500000")

  In this example, the system recognizes as valid both sides of the comparison, but again, the numeric value 500000 is treated as a literal string and compared with the value of the **Credit Limit** field that is returned. Property values are typeless, so the operation of the comparison depends on the expression on the other side. In this case, "500000" is a literal string value (as indicated by the quotation marks), so the property value is compared to it alphabetically as a string and may or may not return a valid result.

- CONDITION(E(MG_CurrentSite) IN ("MI";"ZZ"))

  The MG_CurrentSite parameter is set for all running events. It is used, as in this example, to make it easier for event actions to exit processing based on the value of the current site. A finish action can be used to make a handler only operate for certain sites.

- CONDITION(P("CreditLimit") > 500000)

In this example, again the current value of the **CreditLimit** property is used, but this time it is compared mathematically as a numeric value to the numeric constant 500000. If the property value cannot be converted numerically (for example, if it contains non-digit characters), a runtime error occurs.

For more information about expressions and the functions used to build them, see "Expression Functions" in the *Guide to the Application Event System*.

Event Action Parameter Functions

Predefined functions can be used, within event action parameters, to build expressions that reference values needed by that action. These values can represent variable values, parameter values, or subactions. For example, the GC function can be invoked to reference a particular event global constant value. Or the SUBSTITUTE function can be used to build an expression that substitutes certain values within a defined string at run time.

All of the following functions are available from the **Select a function** drop-down list on the **Event Action Expression Editor** form. You can also use them by typing them manually into an expression on the **Event Actions** form, in the **Parameters** text field.

- ACTIONSEQ
- ACTIONTYPENAME
- ANYHANDLERSFAILED
- APPNAME
- ATTACHMENTEMBEDDED
- ATTACHMENTLIST
- ATTACHMENTNAME
- ATTACHMENTS
- ATTACHMENTSEQ
- BEGINDATE
- BODNOUN
- BODVERB
- BODXML
- CAST
- CEILING
- CLIENTSUBSTITUTE
- COMPANYNAME
- CONFIGNAME
- CURDATETIME
- CUSTOMDELETE
- CUSTOMINSERT
- CUSTOMUPDATE
- DOCDESC
- DOCEXT
- DOCINTERNAL
- DOCMEDIATYPE
- DOCMODIFIED
- DOCNAME
- DOCREADONLY
- DOCSEQ
- DOCTYPE
- E
- EVENTNAME
- EVENTPARMID
- EVENTREVISION
- EVENTSTATE
- EVENTSTATEID
- EVENTTITLE
- FE
- FGC
- FILECONTENTS
- FILTER
- FILTERMETHODPARM
- FILTERPROPERTY
- HANDLERACCESSAS
- HANDLERIGNORESFAILURE
- HANDLERSEQ
- HANDLERSUSPENDS
- HANDLERSYNCHRONOUS
- HANDLERTRANSACTIONAL
- HASBEGUN
- HASFINISHED
- IDO
- IF
- INSIDEDATABASE
- INITIATOR
- INSTR
- LEN
- LOADFLAGS
- LOWER
- MESSAGE
- METHOD
- METHODPARM
- METHODPARMS
- PROPERTYAVAILABLE
- PROPERTYMODIFIED
- PROPERTYNAMES
- RECIPIENTLIST
- RECIPIENTS
- RECORDCAP
- REPLACE
- RESPONDERLIST
- RESPONDERS
- ROUND
- ROWS (for IDO Load Collection action)
- ROWS (for IdoPostLoad-Collection event)
- SUBSTITUTE
- SUBSTRING
- SUBXML
- SV
- TRUNC
- UPPER
- USERDESC

- DATE
- DATEADD
- DATEDIFF
- DATEPART
- DBFUNCTION
- DOCBLOB
- FILTERSTRING
- FLOOR
- FP
- FSV
- FV
- GC
- NEWGUID
- NONRESPONDERLIST
- ORIGINATOR
- P
- POSTQUERYACTIONS
- POWER
- PROPERTY
- USERNAME
- V
- VARIABLENAME
- VARIABLEVALUE
- VOTINGDISPARITY
- VOTINGRESULT
- VOTINGTIE
- WORKINGDIR
- XML2 / XML3 / XML6

In addtion, there are two "pre-parser" functions that you can use when you need an expression that contains elements for other functions/expressions. These are:

- TGC
- TV

For more information about these two functions, see "Pre-parser Functions" in the *Guide to the Application Event System*.

Using Filter Functions

Filter functions are basically used to retrieve a value from somewhere and return it as a string enclosed in single quotation marks. This applies to all variations of filter functions, which include the following:

- FE
- FGC
- FILTER
- FILTERMETHODPARM
- FILTERPROPERTY
- FP
- FSV
- FV

The sole purpose of the filter functions is to construct phrases for SQL WHERE-clauses, which are used in the FILTER() keyword. So it follows that the only use of the standalone FILTER() function is when constructing a phrase for a SQL WHERE-clause using an expression that is not a PROPERTY, V (variable), E (event parameter), SV (session variable), GC (global constant), or METHODPARM (method parameter), because all of those have corresponding filter functions that you could use without bothering about FILTER() itself.

For example, suppose a parameter named **Prefix** is passed to your event. This event is designed to do something to all the items whose item ID code begins with the prefix. You could load those items using a Load IDO Collection action with the following parameters:

```
IDO("SLItems")
FILTER( SUBSTITUTE("Item LIKE {0}", FILTER( E(Prefix) + "*" ) ) )
PROPERTIES("Item, Description")
```

Now suppose your event fires and is passed a prefix of **AL**.

The expression inside the first FILTER() keyword works like this (from the inside out):

- `E(Prefix) + "*"` evaluates to: **AL***
- The `FILTER()` around that places single quotes around it: **'AL*'**
- The `SUBSTITUTE()` function turns that into: **"Item LIKE 'AL*' "**

This resulting string is an ideal SQL WHERE-clause, because the IDO Runtime Service turns the asterisk into a percent-sign that SQL Server understands.

NOTE: On the other hand, an FE(Prefix) expression would evaluate to **" 'AL' "**, which is not conducive to getting the asterisk inside the single-quotes where we need it. So, in effect, we are postponing the wrapping of the single-quotes until just the right time.

The same effect could be implemented using another level of SUBSTITUTE(), or string concatenations using quoted quotes, but FILTER() is cleaner.

Showing Event Action Contexts

When creating event handlers that send system messages to recipients, you might want to make the record for which the event was generated available to the message recipients, in context on its form.

EXAMPLE: Suppose you are setting up an event handler that generates a request for approval to a purchasing manager every time a purchase order is created. To make it easier for the purchasing manager to view the actual request, you can set up the message so that the manager can simply click the **Show Context** button in the **Inbox** form. Clicking this button opens the **Purchase Orders** form with the new purchase order record automatically displayed.

NOTE: The following procedure can be performed only for Notify or Prompt actions, that is, on the **Event Action Notify** or **Event Action Prompt** forms.

To set up a system message to display an associated record in context:

1. Create an event action for a Notify or Prompt action type.
2. By means of the **Edit Parameters** button (on the **Event Actions** form), open the **Event Action Notify** or **Event Action Prompt** form.
3. Click the **Filter Form** button.
4. Use the **Event Action Expression Editor** form to select the FILTERFORM function and designate the name of a form as its argument. Then click **OK**.

   This argument designates the form you want to open when the recipient clicks the **Show Context** button.

   This parameter enables the **Show Context** button when the recipient views the message.
5. Click the **Filter Spec** button.

6 Use the **Event Action Expression Editor** form to select the FILTER function and designate the property on which you want that form to filter when it opens.

This expression will typically take a form similar to the following:

```
FILTER(SUBSTITUTE("propertyName = {0}", FP("propertyName")))
```

This statement allows the system to pass the specific record that triggered the event handler.

7 Set up the rest of the Notify/Prompt action as desired.

Using Custom Entry Forms

For most Prompt messages (that is, messages that require a response from the recipients), you can gather recipients' responses using the **Question** and **Choices** fields. In some cases, though, you might need to gather more specific or detailed responses, or for whatever other reason, you might want to collect recipients' responses by means of a custom form.

To use a custom entry form with a Prompt message:

1 Create the custom entry form.

We recommend that you:

- Create the form from scratch. (See Creating Forms from Scratch.)
- Make sure you include buttons or other devices to allow the recipient to indicate the desired choice.

Use form event handlers to define how these buttons behave. To return the recipient's choice, each of your form's buttons should generate an event, for example, "Accept", with two handlers. The first handler, of type "set values," should set the variable StdVote to the positional number of the voting result, for example 1, or 2, or 3. The second handler should generate the event StdFormExitOk. You should also include a button to allow the recpient to exit without voting. This button should generate an event, for example, "Cancel", with a single handler that generates the event StdFormExitCancel.

2 Include any display fields that you might need to display relevant information.

To display an event variable in a component, bind it to a form variable named for that event variable. For example, to display the event variable "Row.CoNum", create an component of type Edit and set its Data Source Binding attribute to "variables.Row.CoNum". At runtime, the component will be automatically displayed, enabled, and/or decorated according to that event variable's Variable Access setting (i.e., Hidden, Read-Only, Writable, or Mandatory).

To display other information contained in the Prompt message, you can similarly bind components to the following form variables:

- StdFrom
- StdTo
- StdSubject
- StdMessage
- StdSent

- StdCC
- StdQuestion
- StdResponseDate
- StdExpiresAfter
- StdExpired
- StdRead
- StdVote (set to the SelectedChoice property for the current message, or -1 if the message is expired)

You can also use this custom form to collect data values, perform calculations, or do anything else you want it to do. You can perform initialization actions that depend on the above form variables by adding handlers on the StdInboxPayloadInitCompleted event.

3 Create an event action for a Prompt action type.

4 By means of the **Edit Parameters** button (on the **Event Actions** form), open the **Event Action Prompt** form.

5 In the **Entry Form** field, type the name of the custom entry form you created.

6 Set up the rest of the Prompt action as desired.

For a more detailed and working scenario similar to this one, see "Appendix A: Sample Scenarios" in the *Guide to the Application Event System*.

NOTE: Some recipients may have the **Send External Prompts** check box selected on the **Users** form. In this case, you must include a note in the body of the prompt e-mail, instructing users to log in to this application to enter the requested data or perform the actions needed on the custom form.

Registering a BOD Template

Before you can see or select a BOD template for use on the **Event Action Send BOD** form, you must register the template in the system.

To register a BOD template in the system:

1 Open the Maintain BOD Templates form.

2 In the Template Name field, specify the name to identify the template.

If you do not specify a name, the system automatically uses the name of the XML file that you download.

3 Optionally, in the Description field, provide a description.

4 Click Upload Template and use the Upload Template dialog box to browse to, select, and open the XML file that contains the BOD template.

5 Save your changes.

The system loads the template and displays the XML code in the **Templates** read-only field.

To make the template available for immediate use, ensure that the **Active** check box is selected. If you do not select this check box, the template is not displayed on the **Templates** list on the **Event Action Send BOD** form.

To clear the **Template** view and start over, click **Clear Template**.

Creating Event Triggers

A *trigger* is a condition that causes an event to fire. Triggers can be based on conditions that are created:

- By user actions, such as saving and closing a form, changing a record, and so forth
- Apart from user actions, such as the passage of time or the result of a database calculation

The **Event Triggers** form is used to set conditions of the second type.

To create an event trigger:

1. Open the **Event Triggers** form.
2. Press **F3**.
3. Select **Actions** > **New**.
4. From the **Event Name** drop-down list, select the event for which you want to define a trigger.

   NOTE: You cannot define a trigger for a framework (**Core**) event.
5. On the Trigger tab, in the **Condition** field, enter the condition for which the event is to fire.

   For more information about defining conditions, see "About Event Triggers" in the *Guide to the Application Event System*.
6. On the Parameters tab, enter the name and values for any event parameters for which you need to pass values to the event handlers when the event fires.

   For more information about event parameters see "Framework Events" in the *Guide to the Application Event System*.
7. Set other options on this form as desired.

   For more information on other form options see the online Help for each field.
8. Save the form.

For more information, see "About Event Triggers" in the *Guide to the Application Event System*.

Deleting Events

If you are certain you no longer need an event and you want to delete it, you can.

NOTE: You can delete an event only if the **Access As** field has the value of the current Access As value, as displayed on the **Access As** form.

To delete an event:

1 Open the **Events** form and select the event you want to delete.

2 Select **Actions** > **Delete**.

3 Save the form.

Modifying Events

Once an event has been created and saved, the only thing you can modify is the event's description. The event name and other attributes are locked.

NOTE: You can modify an event's description only if the **Access As** field has the value of the current Access As value, as displayed on the **Access As** form.

To modify an event's description:

1 Open the **Events** form and select the event you want to modify.

2 In the **Description** field, modify the description text as desired.

3 Save the form.

Moving Messages Between Folders

You can use the **Saved Messages** form to move messages from one folder to another. If the folder does not already exist, you can create the folder at the same time.

To move a message from one folder to another folder:

1 Open the **Saved Messages** form.

2 From the **Folder Name** drop-down list, select the folder in which the message you want to move is currently placed. The system displays in the grid view all messages in that folder.

3 In the grid view, select the message you want to move.

4 In the **Folder Name** field, do one of the following actions:

- For an existing folder, type in the name of the folder or select the folder from the drop-down list.
- To create a new folder and move the message to that folder, type in the name of the new folder.

5 Save the message.

For more information, see the following topics in the *Guide to the Application Event System*:

- "Event Messages"
- "About the Saved Messages Form"

Resequencing Event Handlers

Use the **Event Handler Sequence** form to change the sequence in which your event handlers execute for a specified event.

NOTES:

- This form is intended to be accessed as a linked form, from the **Event Handlers** form only, using the **Resequence** button. If you open it in standalone mode, the results can be unpredictable.
- You can change the order only of event handlers that have the same Access As value as displayed on the **Access As** form, and *then* only if they are grouped together (that is, adjacent to one another in the sequence). You cannot use this form to change the sequence of event handlers with other Access As values.
- To change the order of your event handlers with respect to those of others (that is, with different Access As values), use the **Keep With** and **Chronology** fields on the **Event Handlers** form.

To change the sequence of an event handler:

1. In the grid view, select an event handler that has your **Access As** value.
2. Click the **Up** or **Down** button to move the selected handler up or down in the sequence, keeping in mind the restrictions mentioned above. If you try to violate those restrictions, the system generates an error and you cannot complete the move.
3. Save the form.

Using the Application Event System for Document (File) Attachments

The application event system provides two basic means to work with document attachments. You can:

- Include documents attached to records when initiating those records for processing using the application event system.
- Use the application event system to attach, link, or detach documents to records during the course of processing in the application event system.

Including Documents Attached to Records

If you plan to use application events and event handlers with collections and records that can already have document attachments, and you want to include those attached documents in whatever action is taking place through the event handling, you can use the Attach event action parameter in conjunction with a Notify, Prompt, or Send E-mail event action. This option allows you to include all documents, only internal documents, all document *other than* internal documents, or specified individual documents from the record being processed. You can also *exclude* all document attachments from an event handler action.

Using the application event system capabilities, you can update (modify) or detach documents already attached to records and included in such actions.

For more information and sample scenarios, see the *Guide to the Application Event System*.

Using the Application Event System to Process Document Attachments

You can use the application event system to attach, link, update, or detach documents during the processing of an event handler. For these types of actions, use the Attach event action type as part of the handler sequence.

For more information and sample scenarios, see the *Guide to the Application Event System*.

Schema (SQL Tables and Columns) Editing

Maintaining Tables and Other SQL Schema Elements

NOTE: Building IDOs over tables in schemas other than dbo is not currently supported.

Creating Tables

To create a SQL table:

1. In the SQL Tables form, select **Actions > New** and specify the table name
2. Verify that the schema is **dbo**.
3. Indicate whether the table includes a multi-site column.
4. Save the record.
5. Click **Columns**.
6. In the **SQL Columns** form, add columns for the new table and define metadata about the columns such as the data type, length, and default value (when applicable to the data type).
7. Save the columns and return to the **SQL Tables** form.
8. Click **New Constraint** to open the **SQL Tables Constraint** form and define one or more primary keys or other constraints for the table, as described below.
9. To save the constraint and return to the **SQL Tables** form, click **OK**.

   After you create tables or columns, you can create IDOs, IDO extension classes, or events that use the tables and columns. You can also filter for a table in the **SQL Tables** form, and alter the columns and other attributes.

   NOTE: Mongoose requires certain columns on tables that it uses. If you import a table into your database, you can filter for it in the **SQL Tables** form, and then click **Update Current Table** to add those required columns.

Maintaining Columns on Tables

Use the **SQL Columns** form to add, delete, or modify columns on tables. However, you cannot make changes to certain restricted tables.

WARNING: Do not delete or modify columns on base application tables. Doing so can cause system instability.

You can also change the definition of an existing column, for example, the data type.

Specifying Primary Keys and Other Constraints for a Table

To define one or more primary keys or other constraints:

1. On the **SQL Tables** form, click one of these buttons:
   - To specify a new constraint, click **New Constraint**. Then specify a **Constraint Type** and related information:
     - **Primary Key:** Specify whether the constraint should be implemented with the clustered attribute.
     - **Index:** Specify whether the constraint is unique (that is, only one unique combination of the columns contained in the constraint is permitted in the table) and whether the constraint should be implemented with the clustered attribute.
     - **Foreign Key:** Specify the name of the table to which the current table refers.
   - To change an existing constraint, select the constraint from the grid and click **Modify Constraint**.

   The data type and constraint name display.
2. Click **Next**.
3. In the left pane, select the column or columns you want to be constraints on the table. To add them to the Keys pane, click **Add**.
4. Click the **Move Up** or **Move Down** buttons to change the order of the columns on the constraint.
5. To delete an existing constraint, remove all columns in the right pane.
6. To save your changes, click **Finish**.

Updating Existing Tables

On the **SQL Tables** form, use the **Update Current Table** button to prepare an existing table for access by Mongoose. Select the table and click the button to perform the following on the table:

- Add the standard Mongoose application columns such as Create Date, Updated By, and so on
- Create the Delete, Insert and Updatepenultimate triggers
- Add application schema table metadata

This is useful when you have a table that you imported into the application database (that is, a table not created with the **SQL Tables** form).

Editing SQL User-Defined Data Types

To create or edit data types from the **SQL Tables** form, click **SQL Data Types** and then:

1. Select Actions > New.
2. Specify the name, base data type, length and precision (if applicable for your base data type) and nullability

3 Save the record.

To change a data type, SQL Server requires that the data type be dropped and recreated. To drop a data type, it must not be in use by a table, stored procedure, or function.

1 Filter for the data type and verify that the **Where Type Used** grid is empty.

2 Select **Actions > Delete** and save the record to drop the data type.

3 Select **Actions > New**, specify the information again, and save the data type record to recreate it.

Executing SQL Statements

From the **SQL Tables** form, click **Apply Database** to display a form where you can execute any SQL statement into the application database. Specify the SQL statement to execute and click **Submit**. If your statement includes a "GO" on a line by itself, everything above it is submitted separately to the database.

The **Apply Database** form must be used only by experts.

SQL Reserved Words

The application reserves the following words that cannot be used in IDO property names or SQL table column names:

add

all

alter

and

any

as

asc

authorization

avg

backup

begin

between

break

browse

bulk

by

cascade

case

check

checkpoint

close

clustered

coalesce

column

commit

committed

compute

confirm

constraint

contains

containstable

continue

controlrow

convert

count

create

cross

current

current_date

current_time

current_timestamp

current_user

cursor

database

dbcc

deallocate

declare

default

delete

deny

desc

disk

distinct

distributed

double

drop

dummy

dump

else

end

errlvl

errorexit

escape

except

exec

execute

exists

exit

fetch

file

fillfactor

floppy

for

foreign

freetext

freetexttable

from

full

goto

grant

group

having

holdlock

identity

identity_insert

identitycol

if

in

index

inner

insert

intersect

into

is

isolation

join

key

kill

left

level

like

lineno

load

max

min

mirrorexit

national

nocheck

nonclustered

not

null

nullif

of

off

offsets

on

once

only

open

opendatasource

openquery

openrowset

option

or

order

outer

over

percent

percision

perm

permanent

pipe

plan

prepare

primary

print

privileges

proc

procedure

processexit

public

raiserror

read

readtext

reconfigure

references

repeatable

replication

restore

restrict

return

revoke

right

rollback

rowcount

rowguidcol

rule

save

schema

select

serializable

session_user

set

setuser

shutdown

some

statistics

sum

system_user

table

tape

temp

temporary

textsize

then

to

top

tran

transaction

trigger

truncate

tsequal

uncommitted

union

unique

update

updatetext

use

user

values

varying

view

waitfor

when

where

while

with

work

writetext

Restricted Tables

Do not add new custom columns to the following tables, and do not extend them with UETs. Customizations to these tables are not preserved during upgrades to the application. An asterisk denotes the table is associated with a form.

ABOPTS_mst

ALTCHG_mst

ALTCHGDTL_mst

ALTERN_mst

** ALTPLAN_mst (Planning Parameters form)

** ALTSCHED_mst (Shop Floor Control Parameters form)

ALTSUM_mst

APPCFG_mst

APSSITE_mst

ATTRIB000_mst

BATCH000_mst

BATPROD000_mst

BATPRODORD000_mst

BATRL000_mst

BATSUM000_mst

BATTIME000_mst

BATWAIT000_mst

BOM000_mst

** CAL000_mst (Holidays form)

CONSPLAN000_mst

DOWN000_mst

DOWNPLAN000_mst

EFFECT000_mst

ERDB_mst

ERDBGW_mst

ERRORLOG_mst

EXRCPT000_mst

FDBVER_mst

FIELDS_mst

GNTHLCAT_mst

GNTHLCRIT_mst

GNTSELCAT_mst

GNTSELMBR_mst

INVPLAN000_mst

JOB000_mst

JOBLNKS000_mst

JOBPLAN000_mst

JOBSTEP000_mst

JS10VR000_mst

JS11VR000_mst

JS12VR000_mst

JS13VR000_mst

JS14VR000_mst

JS15VR000_mst

JS16VR000_mst

JS17VR000_mst

JS18VR000_mst

JS19VR000_mst

JS2VR000_mst

JS3VR000_mst

JS4VR000_mst

JS6VR000_mst

JS7VR000_mst

JS8VR000_mst

JS9VR000_mst

JSATTR000_mst

LOADPERF000_mst

LOADSUM000_mst

LOOKUP000_mst

LSTATUS000_mst

MATADDQ000_mst

MATDELOUT000_mst

MATL000_mst

MATLALT000_mst

MATLATTR000_mst

MATLDELV000_mst

MATLGRP000_mst

MATLPBOMS000_mst

MATLPLAN000_mst

MATLPPS000_mst

MATLRULE000_mst
MATLWHSE000_mst
MATREMQ000_mst
MATSCHD000_mst
MATSUM000_mst
MSLPLAN000_mst
OPRULE000_mst
ORDATTR000_mst
ORDER000_mst
ORDGRP000_mst
ORDIND000_mst
ORDPERF000_mst
ORDPLAN000_mst
ORDSUM000_mst
OSMATL000_mst
PART000_mst
PARTSUM000_mst
PBOM000_mst
PBOMMATLS000_mst
PLANINT000_mst
PLANMSGS000_mst
POEXCEPT000_mst
POLSCHD000_mst
POOL000_mst
POOLQ000_mst
POOLSUM000_mst
PROBDEF_mst
PROCPLN000_mst
RELRECS_mst
REPPAR_mst
RESATTR000_mst
RESLOAD000_mst
RESMNT000_mst
RESPAIR000_mst
RESPLAN000_mst

RESQ000_mst

** RESRC000_mst (Resources form)

RESSCHD000_mst

RESSEND000_mst

RESSUM000_mst

RGATTR000_mst

RGLOAD000_mst

** RGRP000_mst (Resource Groups form)

** RGRPMBR000_mst (Resource Groups form)

RGRPSUM000_mst

SCHEDOP000_mst

** SHIFT000_mst (Scheduling Shifts form)

** SHIFTEXDI000_mst (Resources form, Shift Exceptions tab)

TBLLIST000_mst

TODEMAND000_mst

TOODP000_mst

TOSUPPLY000_mst

TRACELOG000_mst

WAIT000_mst

WHSE000_mst

User Extended Tables (UETs)

User Extended Tables Overview

The User Extended Tables (UET) feature gives the system administrator the ability to extend existing application database tables and add custom user fields to forms in the application. Use this feature to keep track of information that is not currently in the application database schema.

NOTE: If you are using replication, you must click the Regenerate Replication Triggers button on the Replication Management form after UETs are changed, added, or deleted.

Tables are a systematic arrangement of data in records and fields for ready reference. The application ships with tables containing predetermined fields. The UET feature allows users to add their own fields to these tables.

Once a table is extended, you can drop user fields into any application form that uses this table.

NOTE: If you bind a new component to a UET on a form that uses a custom load method stored procedure, an error message displays when you refresh the form. However, the error does not prevent you from continuing.

Only the Primary table of the Form's Primary Collection is extendible on that form. To find the Primary Collection:

1. Select **Form Definition** from the Edit menu.
2. Select **Form Properties** from the Edit Menu.
3. On the Primary tab, click the Collections button.

When data is entered into pre-existing fields, and if the rule expressions you defined for those fields are true, the events to arrange and display information in new custom user fields are triggered.

Follow these steps to add new user fields to forms:

1. Creating a user class - The user class definition is the highest level to extend an application database table.
2. Creating user fields - User fields are generic and can be a part of many classes. If the user changes any property of a user field, all user classes inherit the change.
3. Associating the user field with a user class - The UET tools look for this association to place the user fields in the form that belongs to the user class.
4. Defining the index for the user class - Users who generate their own reports or browse through the classes can take advantage of using an index. This gives users the ability to define their own sorting process in reports. You do not need to define the index for a class, but if you do not, and you sort these fields in custom reports, performance is slowed.

5 Linking an application database table with the user class - The association between a table and a class provides the information that UET needs to retrieve, arrange, and display the user fields that belong to a user class. To link the table with the class, define a rule that determines if the record accessed has a valid user class associated with it. If valid data is entered in existing fields to make the rule expression true, the new user field displays.

6 Impacting the schema - Use the **UET Impact Schema** form to apply the changes you made in the previous steps to all affected databases. This step also updates the corresponding views over multi-site tables.

7 Drawing user fields on application forms - Draw the user fields on the forms that have extended tables associated with them. When the user fields are placed on the form, they act as any other existing field.

User Extended Tables Reports

The following reports are available to view information about UET user classes, user fields, user indexes, and user tables.

- Quick Dictionary Report
- User Class Report
- User Fields Report
- User Index Report

Associating User Fields with a User Class

To associate user fields with a user class:

1 On the **UET Class/Field Relationships** form, select a class in the Class Name field.

2 In the Field Name field, select a field name to associate with the class.

3 Click **OK** to save the relationships and close the form.

Extending Application Database Tables

Follow these step to add custom user fields to application forms by extending application database tables:

1 Create a user class.

2 Create the user fields.

3 Associate the user fields with the user class.

4 Define the index for the class.

5 Create a relationship between an application database table and the user class.

6 Impact the schema.

7 Draw the user fields on forms.

Copying a UET User Field

To copy a UET user field:

1 Select a user field to copy on the UET User Fields form.

2 Click **Copy Field**. The **UET Copy Field** form displays.

3 In the Table field, select a table for the new user field.

4 In the Column field, select a column for the new user field.

5 Click **OK**. You return to the **UET User Field** form.

6 Rename the new user field.

7 If desired, change the attributes for the new user field.

Creating a Relationship Between a Database Table and a User Class

Determining the Primary Table Name for a Form

1 Open the form to which you want to add a UET field.

2 On the **Edit** menu, select **Design Mode**.

3 If the form properties sheet is not open, select **Form Properties** on the **View** menu.

4 Select the **Collections** tab on the property sheet.

5 In the collections tree, select the primary collection.

6 The **Base Table and Alias** property shows the primary table name.

NOTE: Certain tables cannot be extended. See the User Extended Tables Overview topic for a list of these tables.

Associating a User Class with an Application Database Table

1 On the UET Table/Class Relationships form, select **New** from the Actions menu.

2 In the Table Name field, select the table, as determined above, with which you want to associate a class.

3 In the Class Name field, select a class to associate with the table.

4 Select **Active** to be able to draw user fields on application forms.

5 Click the Rule Assistant button to access the **UET Rule Assistant** form.

6 On the **UET Rule Assistant** form, select a field from the (Field) field.

7 In the (Operator) field, select an operator.

8 In the (Value) field, enter a value for the rule.

9 Select **OR instead of AND with previous clause** if you want to find all records that match one of the search criteria instead of finding all records that match the search criteria in the previous clause.

10 Click **Add** to add the rule. The rule displays in the (Criteria) field.

11 Click **OK**. You return to the **UET Table/Class Relationships** form.

12 On the **Form** menu, select **Close and Save Changes**.

Creating a User Class

To create a user class:

1 On the **UET Classes** form, select **New** from the Actions menu.

2 In the **Class Name** field, enter a name to define the class.

3 In the **Label field**, enter a label for the class.

4 In the **Description** field, enter a description of the class.

5 Save the class.

Creating User Fields

To create a user field:

1 On the UET User Fields form, select Actions > New.

2 In the User Field Name field, enter a unique name, prefixed with "uf", to define a user field.

NOTE: The"uf" prefix is not case sensitive, so you can enter something like Uf_Category , uf_Category, or UF_Category as a user field name.

3 In the User Data Type field, select a user data type to define the new user field.

4 In the Data Type field, select a standard SQL Server data type for the user field.

5 In the Precision field, enter the number of alphanumeric characters that can be entered in the user field.

6 In the Decimals field, if you selected Decimal in the Data Type field, enter the number of decimal places available in the user field.

7 In the Initial Value field, enter the initial value you wish to display in the user field.

8 In the **Description** field, enter a description of the user field.

9 Save the user field.

Defining an Index for a Class

To define an index for a class:

1 On the UET Class/Index Relationships form, select **New** from the Actions menu.

2 In the **Class Name** field, select the name of the class you want to index.

3 In the **Index Name** field, enter a unique name for the index.

4 In the **Description** field, enter a description of the index.

5 In the **Field Name** field, select a user field you want to include in the index.

6 Select **Ascending** if you want to sort the fields in the index in ascending order.

7 Save the record.

Drawing UET Fields on Forms

To draw UET fields on forms:

1 Open the form to which you want to add a UET field.

2 Select **Edit > Design Mode**.

3 If the Toolbox is not open, select **View>Toolbox**.

4 In the Toolbox, click the button that corresponds to the user field you want to create.

5 Click on the form where you want one corner of the field to be and drag diagonally until the field is the desired size.

6 In the **Component** properties sheet, specify the properties of the user field.

NOTE: The binding name for a UET user field is **object.<Table-Alias><User-Field-Name>**. The Table-Alias is the three-character abbreviation for a table name.

7 Select Form > Definition > Save.

Impacting the Schema

1. On the UET Impact Schema form, select **Commit Form Changes** to save the changes you have made to the form information using the UET forms.
2. Select **Impact Schema** to change the database schema to contain the columns and indexes corresponding to the user fields you have defined in the UET forms. Corresponding views over multi-site tables are also updated.
3. Click **Process**.

   If your system is multi-site and the table where you add the UET, or the corresponding _all table, is replicating data, there are additional steps you must take. The *Replication Reference Guide*, which is available on our support site, contains a chapter on "Setting Up Custom Replication." In it you can find information about copying table schema changes to other sites.

Critical Numbers

About Critical Numbers

Critical numbers are key performance indicators, or KPIs, that you can use to track your progress. Use critical numbers to answer these questions:

- How am I doing?
- What should I be doing?

Critical numbers are based on a stored procedure calculation or on an IDO calculation. With a license for critical numbers, you can add critical numbers to forms. The system is designed to support both simple and complex critical numbers to meet a wide range of users and needs.

Within the IDO critical number definitions, the user can quickly generate numbers based on things such as:

- The sum, count, average, minimum or maximum of a value
- The **Group By** and **Date Range** properties
- Other properties on the form

You can see the details behind the critical number on an enhanced drilldown screen, and do further analysis on issues you uncover.

Goal and Alert Settings

Critical numbers are color-coded to show their goal and alert status. These colors are used by default:

- **Red**: The critical number is past the alert range.
- **Yellow**: The critical number is between the alert and goal ranges.
- **Green**: The critical number is within the goal range.

Creating a Critical Number

Critical numbers help users track how they are doing. Critical numbers are based on a stored procedure calculation or on an IDO calculation. Custom critical numbers inherit the same security restrictions as the preconfigured numbers.

To build a critical number:

4 Open the **Critical Numbers Setup** form and turn off Filter-in-Place.

5 Select **Actions > New** to create a critical number from scratch, or filter and copy an existing critical number.

6 Specify these fields:

Description

Specify a description for the critical number.

Short Description

Specify a shorter description of the critical number to use as the subject for e-mails relating to it.

Active

Select this check box to actively run the critical number.

Snapshot

Select this check box to keep a snapshot history of the critical number.

Result Divisor

Specify a value if the critical number returns a large value and you want to divide it before it is displayed. For example: 120,000,000 can be displayed as 120 by using a 1,000,000 divisor, and the Critical Number Description could be changed to include the description "(in Millions)".

7 In the Source Type field, select IDO or Stored Procedure.

To build the critical number from an IDO, click **Source Name** to launch the **Critical Number IDO Source Setup** modal form, and specify this information:

IDO Name

Select the IDO that hosts the value you need for your calculation.

Function

Choose the type of calculation to perform. For all functions except Count, you must select the property that is monitored.

- **Sum**: total of all values of the IDO property that meet the filter criteria
- **Count**: number of records that meet the filter criteria
- **Average**: total of all values of the IDO property that meet the filter criteria, divided by the number of records considered
- **Minimum**: the smallest value of the IDO property for all records evaluated
- **Maximum**: the greatest value of the IDO property for all records evaluated

Filter

Click this button if a filter is needed, and follow the instructions outlined in Setting Up a Critical Number/Drilldown IDO Filter.

Display Blank Results

Select this check box to include data with a zero actual value.

Group By

Select a value if you want to group data by a variety of fields.

Date Ranges

Use this section for critical numbers that are date specific. Use the check boxes to select the ranges to be considered.

Date Property

Select a value if only information that falls within a certain time period should be considered.

Description Suffix

Specify a description to distinguish each date specific critical number.

Goal and Alert

Optionally, specify Goal and Alert values to override the values set up on the Critical Numbers Setup form.

If you want to build the critical number from a stored procedure:

Follow the instructions in Stored Procedure Critical Number Examples to create a SQL stored procedure.

Source Name

Specify the name of the stored procedure you want to call.

Source Parms

Optionally, specify a list of values to be passed to the stored procedure. The value of the parameter can change the result of the critical number without altering the critical number code.

8 The **Calculation Definition** describes how the number is calculated. This field is helpful for users who did not create the critical number, and want an overview of the calculation.

9 Specify the **Alert** settings:

Use Alert

Select this check box to use an alert with the critical number.

Alert Operator

Select an operator to use when comparing the Alert Value to the Actual Value.

Alert Value

Specify a value to compare to the Actual Value.

10 Specify the Goal settings:

Use Goal

Select this check box to use a goal with the critical number.

Goal Operator

Select an operator to use when comparing the Goal Value to the Actual Value.

Goal Value

Specify a value to compare to the Actual Value.

11 If you want the critical number to be displayed on the form, specify a **Category** for the critical number on the **Categories** tab. This determines where the critical number is displayed. If you do not want the critical number to be displayed on the form, leave this field blank.

12 Optionally, specify these user permissions on the **Users** tab:

User Name

Specify or select the user name.

Can Drilldown

Select this check box to allow the user to create drilldowns.

Can Run

Select this check box to allow the user to see the selected critical number.

Change Parameters

Select this check box to allow the user to update parameters.

Change Users

Select this check box to allow the user to maintain user permissions for the selected critical number.

Receive Drill Down Email

Select this check box to allow the user to receive drilldown details in e-mails.

Receive Email

Select this check box to allow the user to receive critical number details in e-mails.

13 Optionally, on the **Static Parameters** tab, specify a **Name** and **Value** to set up user-defined values so end users can change critical number values without modifying the source.

14 Optionally, on the **Input Parameters** tab, maintain a list of filters that the critical number was programmed to handle. This data provides readable labels for the stored procedure parameters throughout the system.

15 On the **Snapshots** tab, view a snapshot history of this critical number. Click **Export** to create an Excel spreadsheet from the data. Click **Critical Number Snapshots** to launch the **Critical Number Snapshots** form.

16 Save the new critical number.

Notes

If you want to build the critical number from an IDO, we recommend that you research the IDO against which you need to perform the calculation.

For example, if the critical number calculates the outstanding balance of customers, you would need to find the form that shows outstanding balance, enter design mode of that form, and determine the IDO and property names.

Setting Up Critical Number Parameters

Use the fields on the **Critical Number Parameters** form to set up parameters and values that are generic to all critical numbers, and global settings:

1 On the **General** tab, specify this information:

Alert, Warning and Goal Symbols

For each type of critical number, select a character to be displayed on any e-mails generated from the system.

Load Batch Size

Specify the maximum number of records to be pulled at once from the database when the system retrieves data for Critical Number Drilldowns and DataViews.

2 On the **Global** tab, specify user definable settings that are used for multiple critical numbers.

If the numbers are specific to a critical number, we recommend that you put those on the individual critical number to which they are related.

Setting Up Multiple Results for One Critical Number

For processes that run through a large amount of data, you can create multiple results for one critical number calculation.

Multiple Results for an IDO-based Critical Number

For IDO-based critical numbers, follow the steps in Creating Critical Numbers for building an IDO-based critical number. To generate multiple results, select values for the **Group By** or **Date Property** fields.

Multiple Results for a Stored Procedure-based Critical Number

Generating multiple results with the stored procedure-based critical numbers is more complex. The preconfigured AR Age critical number is an example of one record that can be set up to create hundreds of other numbers, when programmed correctly.

To set up multiple results for a stored procedure-based critical number:

1 Follow the steps in Creating Critical Numbers for building a stored procedure-based critical number.

Note: Consider using static parameters to set up the **Goal Value** and **Alert Value**, and other settings for each record you want to create. For example, the AR Age number uses Alert-1, Alert-2, Alert-3….Alert-7, Goal-1, Goal-2, Goal-3…Goal-7, Bucket-1, Bucket-2…Bucket-7. This ensures that the values are not hard-coded and the end-user can change them without modifying the stored procedure.

2 Write your routine as you do others, but instead of assigning only @Actual, you must create all the #tt_cr_nums records that you want to include. Call the standard procedure SSSWBLoadCrAddSp to accomplish this:

```
CREATE PROCEDURE dbo.WBLoadCrAddSp (
  @KPINum      WBKPINumType
, @Category    WBCategoryType
, @Id          nvarchar(500)
, @Amount      AmountType
, @Description NVARCHAR(500)
, @GoalVal     AmountType = NULL
, @AlertVal    AmountType = NULL
, @MessageTxt  Infobar = NULL
, @GoalOper    WBOperatorType = NULL
, @AlertOper   WBOperatorType = NULL
) AS
```

Notes

Because the results have the same critical number, set the Id to distinguish the numbers. The Id is passed to the drilldown so you can view the appropriate information.

GoalVal and AlertVal are optional. The system uses the values from the **Critical Number Setup** form if you do not override them.

Stored Procedure Critical Number Examples

Setting up a program to create a simple critical number is relatively easy.

The first step is to set up your critical number in the **Critical Numbers** form.

1 Choose a program name for your custom stored procedure. We recommend that you create a naming convention for your custom procedures so they do not conflict with current or future procedures; using a prefix that includes your company name is one way to do this.

2 Choose which parameters you want to be able to set without changing the values in the code. Careful planning of parameter definitions can make the same program usable for multiple critical numbers you want to retrieve. For example, you can set a specific buyer's id or make the same program run against three different warehouses.

3 Keep in mind that you can override any and all settings on this form in your code.

Next, open Query Analyzer or your preferred editor and create your custom stored procedure. The parameters of every critical number are the same and need to look like this (where SSSWBCanCoBookSp is the name of your procedure):

```
CREATE PROCEDURE [dbo].[SSSWBCanCoBookSp] (
```

```
  @KPINum    WBKPINumType
, @AsOfDate DateType
, @Amount   AmountType OUTPUT
, @Parm1           WBSourceNameType = NULL
, @Parm2           WBSourceNameType = NULL
, @Parm3           WBSourceNameType = NULL
, @Parm4           WBSourceNameType = NULL
, @Parm5           WBSourceNameType = NULL
, @Parm6           WBSourceNameType = NULL
, @Parm7           WBSourceNameType = NULL
, @Parm8           WBSourceNameType = NULL
, @Parm9           WBSourceNameType = NULL
, @Parm10          WBSourceNameType = NULL
, @Parm11          WBSourceNameType = NULL
, @Parm12          WBSourceNameType = NULL
, @Parm13          WBSourceNameType = NULL
, @Parm14          WBSourceNameType = NULL
, @Parm15          WBSourceNameType = NULL
, @Parm16          WBSourceNameType = NULL
, @Parm17          WBSourceNameType = NULL
, @Parm18          WBSourceNameType = NULL
, @Parm19          WBSourceNameType = NULL
, @Parm20          WBSourceNameType = NULL
, @Parm21          WBSourceNameType = NULL
, @Parm22          WBSourceNameType = NULL
, @Parm23          WBSourceNameType = NULL
, @Parm24          WBSourceNameType = NULL
, @Parm25          WBSourceNameType = NULL
, @Parm26          WBSourceNameType = NULL
, @Parm27          WBSourceNameType = NULL
, @Parm28          WBSourceNameType = NULL
, @Parm29          WBSourceNameType = NULL
, @Parm30          WBSourceNameType = NULL
, @Parm31          WBSourceNameType = NULL
, @Parm32          WBSourceNameType = NULL
```

```
, @Parm33          WBSourceNameType = NULL
, @Parm34          WBSourceNameType = NULL
, @Parm35          WBSourceNameType = NULL
, @Parm36          WBSourceNameType = NULL
, @Parm37          WBSourceNameType = NULL
, @Parm38          WBSourceNameType = NULL
, @Parm39          WBSourceNameType = NULL
, @Parm40          WBSourceNameType = NULL
, @Parm41          WBSourceNameType = NULL
, @Parm42          WBSourceNameType = NULL
, @Parm43          WBSourceNameType = NULL
, @Parm44          WBSourceNameType = NULL
, @Parm45          WBSourceNameType = NULL
, @Parm46          WBSourceNameType = NULL
, @Parm47          WBSourceNameType = NULL
, @Parm48          WBSourceNameType = NULL
, @Parm49          WBSourceNameType = NULL
, @Parm50          WBSourceNameType = NULL
) AS
```

Write the logic to calculate your value and assign it to @Amount. The amount is returned to be displayed to the user.

In order to retrieve any parameters that you may have set up, you can call a standard function. It is dbo.SSSWBGetParm. Pass in the critical number you are dealing with (@KPINum), and the parameter you want to retrieve. The parameter is looked for first in that specific critical number, and then in the general listing on the **Critical Number Parameters** form. To retrieve a parameter called "Acct" and set it into a variable in your stored procedure, do the following:

SET @Acct = dbo.SSSWBGetParm(@KPINum, 'Acct')

Past Due Order Lines Example:

```
CREATE PROCEDURE SSSWBCanCoitemPastDueSp (
  @KPINum           WBKPINumType
, @AsOfDate         DateType
, @Amount           AmountType OUTPUT
, @Parm1            WBSourceNameType = NULL
, @Parm2            WBSourceNameType = NULL
, @Parm3            WBSourceNameType = NULL
```

```
, @Parm4          WBSourceNameType = NULL
, @Parm5          WBSourceNameType = NULL
, @Parm6          WBSourceNameType = NULL
, @Parm7          WBSourceNameType = NULL
, @Parm8          WBSourceNameType = NULL
, @Parm9          WBSourceNameType = NULL
, @Parm10         WBSourceNameType = NULL
, @Parm11         WBSourceNameType = NULL
, @Parm12         WBSourceNameType = NULL
, @Parm13         WBSourceNameType = NULL
, @Parm14         WBSourceNameType = NULL
, @Parm15         WBSourceNameType = NULL
, @Parm16         WBSourceNameType = NULL
, @Parm17         WBSourceNameType = NULL
, @Parm18         WBSourceNameType = NULL
, @Parm19         WBSourceNameType = NULL
, @Parm20         WBSourceNameType = NULL
, @Parm21         WBSourceNameType = NULL
, @Parm22         WBSourceNameType = NULL
, @Parm23         WBSourceNameType = NULL
, @Parm24         WBSourceNameType = NULL
, @Parm25         WBSourceNameType = NULL
, @Parm26         WBSourceNameType = NULL
, @Parm27         WBSourceNameType = NULL
, @Parm28         WBSourceNameType = NULL
, @Parm29         WBSourceNameType = NULL
, @Parm30         WBSourceNameType = NULL
, @Parm31         WBSourceNameType = NULL
, @Parm32         WBSourceNameType = NULL
, @Parm33         WBSourceNameType = NULL
, @Parm34         WBSourceNameType = NULL
, @Parm35         WBSourceNameType = NULL
, @Parm36         WBSourceNameType = NULL
, @Parm37         WBSourceNameType = NULL
, @Parm38         WBSourceNameType = NULL
```

```
, @Parm39          WBSourceNameType = NULL
, @Parm40          WBSourceNameType = NULL
, @Parm41          WBSourceNameType = NULL
, @Parm42          WBSourceNameType = NULL
, @Parm43          WBSourceNameType = NULL
, @Parm44          WBSourceNameType = NULL
, @Parm45          WBSourceNameType = NULL
, @Parm46          WBSourceNameType = NULL
, @Parm47          WBSourceNameType = NULL
, @Parm48          WBSourceNameType = NULL
, @Parm49          WBSourceNameType = NULL
, @Parm50          WBSourceNameType = NULL
) AS
DECLARE @CoStatList  LongListType
, @CoitemStatList    LongListType
, @CredHold          ListYesNoType
, @LateDays          GenericIntType
, @QtyDue            QtyUnitType
, @OrdTotal          AmountType
, @ParmsSite         SiteType
, @CustNum           CustNumType
, @CoNum             CoNumType
, @LineFilter        CoLineType
, @ItemFilter        ItemType
, @ProdCodeFilter    ProductCodeType
, @WhseFilter        WhseType
, @StatFilter        CoitemStatusType
SELECT @ParmsSite = site
FROM parms
SET @CoStatList     = ISNULL(dbo.SSSWBGetParm(@CrNum, 'COStatusList'), 'POS')
SET @CoitemStatList = ISNULL(dbo.SSSWBGetParm(@CrNum, 'COITEMStatusList'), 'PO')
SET @CredHold       = ISNULL(dbo.SSSWBGetParm(@CrNum, 'CredHold'), 0)
SET @LateDays       = ISNULL(dbo.SSSWBGetParm(@CrNum, 'LateDaysTolerance'), 0)
SET @CustNum        = dbo.ExpandKyByType('CustNumType', @Parm1)
SET @CoNum          = dbo.ExpandKyByType('CoNumType', @Parm2)
```

```
SET @LineFilter = NULLIF(@Parm3, '')
SET @ItemFilter = NULLIF(@Parm4, '')
SET @ProdCodeFilter = NULLIF(@Parm5, '')
SET @WhseFilter = NULLIF(@Parm6, '')
SET @StatFilter = NULLIF(@Parm7, '')
SELECT @Amount = COUNT(*)
FROM coitem
LEFT OUTER JOIN item itm
  ON itm.item = coitem.item
WHERE (@CoNum IS NULL OR co_num = @CoNum)
  AND charindex(coitem.stat, @CoitemStatList) > 0
  AND qty_ordered > qty_shipped
  AND ISNULL(due_date, '1900-01-01') &GT= dateadd(dd, @LateDays, @AsOfDate)
  AND ship_site = @ParmsSite
  AND EXISTS (SELECT 1 FROM co
              WHERE (@CustNum IS NULL OR co.cust_num = @CustNum)
                AND co.co_num = coitem.co_num
                AND charindex(co.stat, @CoStatList) > 0
                AND co.credit_hold = @CredHold
             )
  AND (@LineFilter IS NULL OR coitem.co_line = @LineFilter)
  AND (@ItemFilter IS NULL OR coitem.item = @ItemFilter)
  AND (@ProdCodeFilter IS NULL OR itm.product_code = @ProdCodeFilter)
  AND EXISTS (SELECT 1 FROM co
              WHERE (@WhseFilter IS NULL OR co.whse = @WhseFilter)
                AND (@StatFilter IS NULL OR co.stat = @StatFilter)
                AND co.co_num = coitem.co_num)

RETURN 0
```

Changing Critical Number Display Settings

Use the **Critical Number Display Settings** modal form to change the display settings of a critical number gauge.

To change the settings:

1 On any critical number gauge, right click and select **Display Settings**.

2 Use these fields to change any settings:

Critical Number

The number of the current gauge is displayed.

Date Range

Select the range of dates to include in the gauge: all dates, year-to-date, or period-to-date.

Group

If the gauge is associated with a group, the group name is displayed.

As of Date

Select the date against which the critical number is calculated.

Drilldown

Select a drilldown for the critical number.

Gauge Type

Select the type of gauge to use:

- Angular Gauge
- Bulb
- Cylinder
- Horizontal LED
- Horizontal Linear Gauge
- Thermometer
- Vertical LED
- Horizontal Bullet
- Vertical Bullet

Goal Value

Specify a value to signify the desired amount. This value is compared to the actual value.

Alert Value

Specify a value to signify an alert when compared to the actual value.

Caption Format

Select the caption format to use for the critical number:

- Long
- Short
- Default
- None (static gauge option)

Sub-Caption Format

Select the sub-caption format to use for the critical number:

- All (goal/alert/actual)
- Actual
- None (static gauge option)

3 Click **OK** to save your changes.

4 Click **Cancel** to close the form without saving.

About Critical Number Snapshots

Critical number snapshots provide a history of critical numbers and show if the numbers are improving. You can set snapshots to run in the background queue or run them any time using the **Snapshot Generation** utility.

You can export the list of snapshots to create charts and graphs using the **Critical Numbers Setup** form.

This information is displayed on the **Critical Number Snapshots** form:

- The critical number category, ID, and description.
- The date that the **Snapshot Generation** utility ran.
- The actual (calculated) value of the critical number.
- **Use Alert** is selected if the critical number uses a color indicator or symbol to show if the actual value is outside the alert range.
- A description of the alert range.
- **Use Goal** is selected if the critical number uses a color indicator or symbol to show if the actual value met the goal.
- A description of the goal value.
- **Symbol** shows how the actual value of the critical number compares to the goal and alert values set up for that number. The possible values are:
  - Goal: The critical number is meeting its goal.
  - Alert: The critical number requires attention.
  - Warning: The critical number is meeting its goal, but might require attention soon.
  - None: The **Use Goal** and **Use Alert** check boxes have been cleared for the critical number.

Using the Snapshot Generation Utility

To generate a critical number snapshot:

1 Launch the **Snapshot Generation** utility.

2 Select an **As of** date to determine when the calculation of the critical number begins.

3 Optionally, select **Append to Current Day** to add the snapshot to an existing list of critical numbers.

If you do not select this check box, the new snapshot replaces any existing snapshots.

4 Click **Process** to run the generation.

Critical Number Drilldowns

About Critical Number Drilldowns

Because a critical number shows only one result value, it is often necessary to access the detailed records that comprise the results. You can configure critical number drilldowns to return a set of data. For example, you can set up a drilldown for the inventory value critical number to view the items in the inventory, and the item warehouse, location, and value.

Typically, the IDO and filter used on the **Critical Numbers Setup** form are the same primary IDO and filter used when defining the drilldown. However, additional tables and IDOs might need to be referenced in the drilldown to provide user-readable output.

Like critical numbers, drilldowns can be configured to use IDOs or stored procedures.

Setting Up a Critical Number Drilldown

To set up an IDO-based drilldown, follow the steps in Setting Up an IDO Critical Number Drilldown. To set up a drilldown based on a stored procedure, follow the steps in Setting Up a Stored Procedure Critical Number Drilldown.

Critical number drilldowns are displayed in a DataView. As with DataViews, you can manipulate the data and save a layout whether you drill into data details from a critical number control or activate the drilldown from the **Critical Numbers** form. See About DataViews for more information on the DataView tool.

Setting Up a Critical Number Drilldown Based on a Stored Procedure

To set up a drilldown based on a stored procedure:

1. Open the **Drilldowns Setup** form and turn off Filter-in-Place.
2. Select **Actions > New** to create a drilldown, or copy an existing drilldown.
3. Specify this information:

   Description

   Specify a description for the drilldown.

NOTE: If the data needs to be refined by a hard-coded list of values, we recommend that you assign the name and value on the **Static Parameters** tab. You can then reference the static parameters when you set up the critical number/drilldown filter using the CRPARM() syntax. For example, the critical number based on the Account Balance stored procedure has a static parameter called Acct. This tells the stored procedure which account number to use. The number that ships with the product uses 10000 (cash), but the user can change this value if they want to monitor a different account or if their cash account uses a different account number.

Source Type

Select **Stored Procedure**.

Source Name

Specify a name for the stored procedure. We recommend that you name the stored procedure the same as the critical number stored procedure with the suffix of "Dtl." For example, Critical Number SSSWBCanCoShipSp would use SSSWBCanCoShipDtlSp.

4 If necessary, specify the **Alert Value** and **Goal Value**. You can use these values to regulate the critical number and track its actual value.

5 On the **Ouput Columns** tab, set the values to be returned in fields in temporary tables.

6 Optionally, select **Use Standard Display**.

7 Optionally, select **Email Drilldown** to include the drilldown in the e-mail notification.

8 Save the record.

9 Associate the drilldown to a critical number:

- Launch the **Critical Numbers Setup** form and navigate to the critical number.
- On the **Drilldowns** tab, select the new drilldown and specify a description.
- Save the record.

10 Create your stored procedure in your preferred editor.

The parameters for a drilldown are slightly different from a critical number, as shown in this example:

```
CREATE PROCEDURE WBCanCoitemPastDueDetailsSp (
  @AsOfDate       DateType
, @DrillNum       WBDrillNumType
, @KPINum         WBKPINumType
, @Id             nvarchar(500)
, @Parm1          WBSourceNameType
, @Parm2          WBSourceNameType
, @Parm3          WBSourceNameType
```

```
, @Parm4        WBSourceNameType
, @Parm5        WBSourceNameType
, @Parm6        WBSourceNameType
, @Parm7        WBSourceNameType
, @Parm8        WBSourceNameType
, @Parm9        WBSourceNameType
, @Parm10       WBSourceNameType
, @Parm11       WBSourceNameType
, @Parm12       WBSourceNameType
, @Parm13       WBSourceNameType
, @Parm14       WBSourceNameType
, @Parm15       WBSourceNameType
, @Parm16       WBSourceNameType
, @Parm17       WBSourceNameType
, @Parm18       WBSourceNameType
, @Parm19       WBSourceNameType
, @Parm20       WBSourceNameType
, @Parm21       WBSourceNameType
, @Parm22       WBSourceNameType
, @Parm23       WBSourceNameType
, @Parm24       WBSourceNameType
, @Parm25       WBSourceNameType
, @Parm26       WBSourceNameType
, @Parm27       WBSourceNameType
, @Parm28       WBSourceNameType
, @Parm29       WBSourceNameType
, @Parm30       WBSourceNameType
, @Parm31       WBSourceNameType
, @Parm32       WBSourceNameType
, @Parm33       WBSourceNameType
, @Parm34       WBSourceNameType
, @Parm35       WBSourceNameType
, @Parm36       WBSourceNameType
, @Parm37       WBSourceNameType
```

```
    , @Parm38       WBSourceNameType
    , @Parm39       WBSourceNameType
    , @Parm40       WBSourceNameType
    , @Parm41       WBSourceNameType
    , @Parm42       WBSourceNameType
    , @Parm43       WBSourceNameType
    , @Parm44       WBSourceNameType
    , @Parm45       WBSourceNameType
    , @Parm46       WBSourceNameType
    , @Parm47       WBSourceNameType
    , @Parm48       WBSourceNameType
    , @Parm49       WBSourceNameType
    , @Parm50       WBSourceNameType
    ) AS
```

11 Records are returned to the user through the WBTmpDrilldowns temporary table. The columns that you set in this temporary table correspond to the columns you set up on the **Output Columns** tab on the **Drilldowns Setup** form. If you specified a Detail form on the **Drilldowns Setup** form and you want to see details on the specific record, set the RowPointer, as shown in this example:

```
INSERT INTO WBTmpDrilldowns(
  RefRowPointer
, DATE01
, CHAR01
, INTE01
, CHAR02
, CHAR03
, SessionID
)
SELECT
  coitem.RowPointer
, coitem.due_date
, co.co_num
, coitem.co_line
```

```
, co.cust_num
, custaddr.name
, @SessionID
FROM coitem
INNER JOIN co
   ON co.co_num = coitem.co_num
LEFT OUTER JOIN custaddr
  ON custaddr.cust_num = co.cust_num
 AND custaddr.cust_seq = co.cust_seq
LEFT OUTER JOIN item itm
  ON itm.item = coitem.item
WHERE co.cust_num = ISNULL(NULLIF(@CustNum,''), co.cust_num)
  AND co.co_num = ISNULL(NULLIF(@CoNum,''), co.co_num)
  AND charindex(coitem.stat, @CoitemStatList) > 0
  AND qty_ordered > qty_shipped
  AND ISNULL(due_date, '1900-01-01') = dateadd(dd,="" @latedays,=""
@asofdate)="" and="" ship_site="@ParmsSite" charindex(co.stat,=""
@costatlist)="">> 0
  AND co.credit_hold = @CredHold
  AND (@LineFilter IS NULL OR coitem.co_line = @LineFilter)
  AND (@ItemFilter IS NULL OR coitem.item = @ItemFilter)
  AND (@ProdCodeFilter IS NULL OR itm.product_code = @ProdCodeFilter)
  AND (@WhseFilter IS NULL OR co.whse = @WhseFilter)
  AND (@StatFilter IS NULL OR co.stat = @StatFilter)
```

12 You can set these additional values in the WBTmpDrilldowns table to affect what is displayed in the drilldown:

- RowPointer provides a link to the specific record when launching a detail form.
- GoalValue overrides the **Goal Value** from the **Drilldowns Setup** form.
- AlertValue overrides the **Alert Value** from the **Drilldowns Setup** form.

13 Associate the drilldown to a category.

NOTE: Critical numbers that do not have one or more categories associated are not displayed on the **Critical Numbers** form.

14 Save the drilldown.

Customer Order Past Due Example

```
CREATE PROCEDURE WBCanCoitemPastDueDetailsSp (
  @AsOfDate        DateType
, @DrillNum        WBDrillNumType
, @KPINum           WBKPINumType
, @Id              nvarchar(500)
, @Parm1           WBSourceNameType
, @Parm2           WBSourceNameType
, @Parm3           WBSourceNameType
, @Parm4           WBSourceNameType
, @Parm5           WBSourceNameType
, @Parm6           WBSourceNameType
, @Parm7           WBSourceNameType
, @Parm8           WBSourceNameType
, @Parm9           WBSourceNameType
, @Parm10          WBSourceNameType
, @Parm11          WBSourceNameType
, @Parm12          WBSourceNameType
, @Parm13          WBSourceNameType
, @Parm14          WBSourceNameType
, @Parm15          WBSourceNameType
, @Parm16          WBSourceNameType
, @Parm17          WBSourceNameType
, @Parm18          WBSourceNameType
, @Parm19          WBSourceNameType
, @Parm20          WBSourceNameType
, @Parm21          WBSourceNameType
, @Parm22          WBSourceNameType
, @Parm23          WBSourceNameType
, @Parm24          WBSourceNameType
, @Parm25          WBSourceNameType
, @Parm26          WBSourceNameType
, @Parm27          WBSourceNameType
, @Parm28          WBSourceNameType
```

```
, @Parm29          WBSourceNameType
, @Parm30          WBSourceNameType
, @Parm31          WBSourceNameType
, @Parm32          WBSourceNameType
, @Parm33          WBSourceNameType
, @Parm34          WBSourceNameType
, @Parm35          WBSourceNameType
, @Parm36          WBSourceNameType
, @Parm37          WBSourceNameType
, @Parm38          WBSourceNameType
, @Parm39          WBSourceNameType
, @Parm40          WBSourceNameType
, @Parm41          WBSourceNameType
, @Parm42          WBSourceNameType
, @Parm43          WBSourceNameType
, @Parm44          WBSourceNameType
, @Parm45          WBSourceNameType
, @Parm46          WBSourceNameType
, @Parm47          WBSourceNameType
, @Parm48          WBSourceNameType
, @Parm49          WBSourceNameType
, @Parm50          WBSourceNameType
) AS
DECLARE @Severity INT
, @CoNum    CoNumType
, @CustNum  CustNumType
, @ParmsSite SiteType
, @CoStatList  LongListType
, @CoitemStatList    LongListType
, @CredHold          ListYesNoType
, @LateDays INT
, @LineFilter        CoLineType
, @ItemFilter        ItemType
, @ProdCodeFilter    ProductCodeType
, @WhseFilter        WhseType
```

```
, @StatFilter        CoitemStatusType

, @SessionID      RowPointerType

SET @Severity = 0

SET @LateDays = 0

SET @SessionId = dbo.SessionIdSp()

SET @CustNum = dbo.ExpandKyByType('CustNumType', @Parm1) SET @CoNum =
dbo.ExpandKyByType('CoNumType', @Parm2) SET @LineFilter = NULLIF(@Parm3, '') SET
@ItemFilter = NULLIF(@Parm4, '') SET @ProdCodeFilter = NULLIF(@Parm5, '') SET
@WhseFilter = NULLIF(@Parm6, '') SET @StatFilter = NULLIF(@Parm7, '')

IF @CoNum IS NULL AND @LineFilter IS NOT NULL

   SET @LineFilter = NULL

SELECT @ParmsSite = site

FROM parms

SET @CoStatList     = ISNULL(dbo.WBGetDrillParm(@DrillNum, @KPINum,
'COStatusList'), 'POS')

SET @CoitemStatList = ISNULL(dbo.WBGetDrillParm(@DrillNum, @KPINum,
'COITEMStatusList'), 'PO')

SET @CredHold       = ISNULL(dbo.WBGetDrillParm(@DrillNum, @KPINum, 'CredHold'),
0)

SET @LateDays       = ISNULL(dbo.WBGetDrillParm(@DrillNum, @KPINum,
'LateDaysTolerance'), 0)

INSERT INTO WBTmpDrilldowns(

  RefRowPointer

, DATE01

, CHAR01

, INTE01

, CHAR02

, CHAR03

, SessionID

)

SELECT

  coitem.RowPointer

, coitem.due_date

, co.co_num

, coitem.co_line

, co.cust_num

, custaddr.name

, @SessionID
```

```
FROM coitem
INNER JOIN co
   ON co.co_num = coitem.co_num
LEFT OUTER JOIN custaddr
  ON custaddr.cust_num = co.cust_num
 AND custaddr.cust_seq = co.cust_seq
LEFT OUTER JOIN item itm
  ON itm.item = coitem.item
WHERE co.cust_num = ISNULL(NULLIF(@CustNum,''), co.cust_num)
  AND co.co_num = ISNULL(NULLIF(@CoNum,''), co.co_num)
  AND charindex(coitem.stat, @CoitemStatList) > 0
  AND qty_ordered > qty_shipped
  AND ISNULL(due_date, '1900-01-01') = dateadd(dd,="" @latedays,="" @asofdate)=""
and="" ship_site="@ParmsSite" charindex(co.stat,="" @costatlist)="" 0
  AND co.credit_hold = @CredHold
  AND (@LineFilter IS NULL OR coitem.co_line = @LineFilter)
  AND (@ItemFilter IS NULL OR coitem.item = @ItemFilter)
  AND (@ProdCodeFilter IS NULL OR itm.product_code = @ProdCodeFilter)
  AND (@WhseFilter IS NULL OR co.whse = @WhseFilter)
  AND (@StatFilter IS NULL OR co.stat = @StatFilter)
RETURN @Severity
```

Setting Up a Critical Number Drilldown Based on an IDO

To set up a drilldown based on an IDO:

1. Open the **Drilldowns Setup** form and turn off Filter-in-Place.
2. Select **Actions > New** to create a drilldown from scratch, or copy an existing drilldown.
3. Specify this information:

   Description

   Specify a description for the drilldown.

   NOTE: If the data needs to be refined by a hard-coded list of values, we recommend that you assign the name and value on the Static Parameters tab. You can then reference the static parameters when you set up the critical number/drilldown filter using the CRPARM() syntax. For example, the Opportunities Neglected Value critical number has a static parameter called DaysNeglected that allows users to control how many days have to pass before an opportunity is considered neglected. It is easier to set a static parameter than it is to

understand this line in the filter and change it: AND (DATEDIFF(d, DerMostRecentCompleteDate,(DATEADD(d, CRPARM('OffsetDays'), CRPARM('AsOfDate'))))) >= CRPARM('DaysNeglected').

Source Type

Select **IDO**.

Source Name

Click this button to launch the **Drilldowns IDO Setup** Form.

4 On the **Drilldown IDO Setup** form, specify this information:

IDO Name

Select the IDO against which you want to build the drilldown.

Filter

If a filter is needed, click this button and follow the directions outlined in Setting Up a Critical Number/Drilldown IDO Filter.

Goal/Alert Property Name

Select the property of the IDO to be evaluated by the goal or alert logic.

Date Property Name

Select a date property if the drilldown detail is for a specific period of time.

Output Columns

Select **Include** for each value that you want to include when the drilldown is displayed.

Input Parameters

Select **Include** for each property of the IDO that can be used to dynamically filter the results of the drilldown.

5 Click **OK** to accept the IDO settings and return to the **Drilldown Setup** form. Click **Cancel** to clear the settings and return to the **Drilldown Setup** form.

NOTE: The properties selected in the grids are automatically added to the grid of the Input Parameter tab of the Drilldown Setup when you click **OK**.

6 If necessary, specify the **Alert Value** and **Goal Value**. You can use these values to regulate the critical number and track its actual value.

7 Optionally, select **Email Drilldown** to include the drilldown in the email notification.

8 Use the tabs to set additional information:

On the **Output Columns** tab, you can arrange the sequence order of the columns.

On the **Categories** tab, you can associate a critical number category to the drilldown.

On the **Sub Drilldowns** tab, you can view any associated sub drilldowns.

Use the parameters tabs to set additional parameters.

9 Save the new record.

10 Follow these steps to associate the drilldown with a critical number:

- Launch the **Critical Numbers Setup** form and navigate to the critical number.
- On the **Drilldowns** tab, select the new drilldown and specify a description.
- Save the record.

Setting Up a Critical Number/Drilldown IDO Filter

To launch the **Critical Number/Drilldown IDO Filter Setup** form, click **Filter** on the **Critical Number IDO Source Setup** form or the **Drilldowns IDO Setup** form. Use filters to filter the query performed against the database to narrow the selected records. For example, you would specify Stat=R to filter customer orders to regular orders.

To build the filter:

1 Select the **Property Name** of the IDO to be evaluated.

2 Select the **Operation Method**. If you select **Is Null** or **Is Not Null**, do not specify a **Comparison Type** or **Comparison Value**.

3 Select the **Comparison Type**:

- **Literal**: The property is compared to the literal value you specify in **Comparison Value**.
- **Property Name**: The property is compared to another property in the same IDO. The other property is specified in the **Comparison Value** field.
- **Critical Number Parameter**: The property is compared to the value of a static parameter defined on the **Critical Numbers** form. This parameter is specified in the **Comparison Value** field.

4 Specify a **Comparison Value**, based on the **Comparison Type** you specified.

5 Click **Add** to translate the information from the fields into filter syntax, and display the filter text in the editor field. Click **Remove** to erase the content in the editor field.

For complex comparison logic, you can edit the text in the editor field to add parentheses around OR and AND statements, to ensure that the filter is evaluated properly.

Click **Cancel** to ignore your changes. Click **OK** to save changes.

NOTE: To better understand how filters can be used, look at the preconfigured critical numbers that are provided as part of the application installation.

Setting Up a Sub-Drilldown

Second level drilldowns can be very useful in cases where a critical number is calculated from sub-totaled data, and the details of the sub-total need to be accessible. You can use Output Column property names of the parent drilldown as input parameters for the sub-drilldown.

Setting up an IDO-based Sub-Drilldown

Follow the steps in Setting Up a Critical Number Drilldown Based on an IDO to create the drilldown. To associate the new drilldown as a sub-drilldown:

1 Launch the **Drilldowns Setup** form.

2 Navigate to the top level drilldown.

3 On the **Sub Drilldowns** tab, specify the new drilldown and a description for it.

4 Save the record.

Setting up a Sub Level Drilldown Based on a Stored Procedure

The preconfigured Inventory Value critical number is a good example of second level drilldowns for stored procedure-based critical numbers. This critical number shows your whole inventory value, and drills down to a subtotal by inventory. It drills down one step further to item totals or location totals by warehouse.

Follow the steps in Setting Up a Stored Procedure-based Critical Number Drilldown to create the drilldown. To associate the new drilldown as a sub drilldown:

5 Optionally, use the @Parms parameters to accept filters into your drilldown.

   For example, the Inventory Value Detail Drilldown (SSSWBCanInvValDtlSp) accepts Whse in @Parm1, Item in @Parm2, and Location in @Parm3. It is coded to use these values as filters if provided, or ignore them if they are not provided.

6 Specify **Source Parms** in sequence for how you want to accept your parameters in your stored procedure.

   When one drilldown calls another, the sub-drilldown automatically pulls these values by column header name from the calling drilldown. For example, the Item Inventory Value Detail drilldown has a Column Heading named Item. When it calls the Inventory Value Detail sub-drilldown, the Item value is passed to the Inventory Value Detail program in @Parm2.

7 Launch the **Drilldowns Setup** form.

8 Navigate to the top level drilldown.

9 On the **Sub Drilldowns** tab, specify the new drilldown and a description for it.

10 Save the record.

First Level Drilldown Program:
SSSWBCanInvValItemDtlSp

```
CREATE PROCEDURE SSSWBCanInvValItemDtlSp (
  @AsOfDate        DateType
, @DrillNum        WBDrillNumType
, @CrNum           WBCrNumType
, @Id              nvarchar(500)
```

```
, @Parm1          WBSourceNameType
, @Parm2          WBSourceNameType
, @Parm3          WBSourceNameType
, @Parm4          WBSourceNameType
, @Parm5          WBSourceNameType
, @Parm6          WBSourceNameType
, @Parm7          WBSourceNameType
, @Parm8          WBSourceNameType
, @Parm9          WBSourceNameType
, @Parm10         WBSourceNameType
, @Parm11         WBSourceNameType
, @Parm12         WBSourceNameType
, @Parm13         WBSourceNameType
, @Parm14         WBSourceNameType
, @Parm15         WBSourceNameType
, @Parm16         WBSourceNameType
, @Parm17         WBSourceNameType
, @Parm18         WBSourceNameType
, @Parm19         WBSourceNameType
, @Parm20         WBSourceNameType
, @Parm21         WBSourceNameType
, @Parm22         WBSourceNameType
, @Parm23         WBSourceNameType
, @Parm24         WBSourceNameType
, @Parm25         WBSourceNameType
, @Parm26         WBSourceNameType
, @Parm27         WBSourceNameType
, @Parm28         WBSourceNameType
, @Parm29         WBSourceNameType
, @Parm30         WBSourceNameType
, @Parm31         WBSourceNameType
, @Parm32         WBSourceNameType
, @Parm33         WBSourceNameType
, @Parm34         WBSourceNameType
, @Parm35         WBSourceNameType
```

```
, @Parm36          WBSourceNameType
, @Parm37          WBSourceNameType
, @Parm38          WBSourceNameType
, @Parm39          WBSourceNameType
, @Parm40          WBSourceNameType
, @Parm41          WBSourceNameType
, @Parm42          WBSourceNameType
, @Parm43          WBSourceNameType
, @Parm44          WBSourceNameType
, @Parm45          WBSourceNameType
, @Parm46          WBSourceNameType
, @Parm47          WBSourceNameType
, @Parm48          WBSourceNameType
, @Parm49          WBSourceNameType
, @Parm50          WBSourceNameType
) AS
DECLARE
  @RowPointer RowPointer
, @TmpAmount  AmountType
, @StartItem  ItemType
, @EndItem    ItemType
DECLARE @ttItemloc TABLE (
  RowPointer uniqueidentifier
, item       nvarchar(30)
, amount     decimal(20,8)
, processed  tinyint
)
SET @StartItem  = ISNULL(@Parm1, dbo.LowString('ItemType'))
SET @EndItem    = ISNULL(@Parm1, dbo.HighString('ItemType'))
INSERT INTO @ttItemloc
SELECT RowPointer, item, 0, 0
FROM itemloc
WHERE item BETWEEN @StartItem AND @EndItem
WHILE EXISTS (SELECT TOP 1 1 FROM @ttItemloc WHERE processed = 0)
BEGIN
```

```
   SELECT TOP 1 @RowPointer = RowPointer
   FROM @ttItemloc
   WHERE processed = 0
   SET @TmpAmount = 0
   EXEC SSSWBCanInvValSubItemlocSp @RowPointer, @TmpAmount OUTPUT
   UPDATE @ttItemloc
   SET amount = @TmpAmount
   , processed = 1
   WHERE RowPointer = @RowPointer
END
INSERT INTO #tt_drill_results(
  CHAR01, DECI01, amount
)
SELECT item, SUM(amount), SUM(amount)
FROM @ttItemloc
GROUP BY item
UPDATE #tt_drill_results
SET RowPointer = item.RowPointer
FROM #tt_drill_results tt, item
WHERE item.item = tt.CHAR01
RETURN 0
```

Second Level Drilldown Program:
SSSWBCanInvValItemDtlSp

```
CREATE PROCEDURE SSSWBCanInvValDtlSp (
  @AsOfDate        DateType
, @DrillNum        WBDrillNumType
, @CrNum           WBCrNumType
, @Id              nvarchar(500)
, @Parm1           WBSourceNameType
, @Parm2           WBSourceNameType
, @Parm3           WBSourceNameType
, @Parm4           WBSourceNameType
, @Parm5           WBSourceNameType
, @Parm6           WBSourceNameType
, @Parm7           WBSourceNameType
```

```
    , @Parm8            WBSourceNameType
    , @Parm9            WBSourceNameType
    , @Parm10           WBSourceNameType
    , @Parm11           WBSourceNameType
    , @Parm12           WBSourceNameType
    , @Parm13           WBSourceNameType
    , @Parm14           WBSourceNameType
    , @Parm15           WBSourceNameType
    , @Parm16           WBSourceNameType
    , @Parm17           WBSourceNameType
    , @Parm18           WBSourceNameType
    , @Parm19           WBSourceNameType
    , @Parm20           WBSourceNameType
    , @Parm21           WBSourceNameType
    , @Parm22           WBSourceNameType
    , @Parm23           WBSourceNameType
    , @Parm24           WBSourceNameType
    , @Parm25           WBSourceNameType
    , @Parm26           WBSourceNameType
    , @Parm27           WBSourceNameType
    , @Parm28           WBSourceNameType
    , @Parm29           WBSourceNameType
    , @Parm30           WBSourceNameType
    , @Parm31           WBSourceNameType
    , @Parm32           WBSourceNameType
    , @Parm33           WBSourceNameType
    , @Parm34           WBSourceNameType
    , @Parm35           WBSourceNameType
    , @Parm36           WBSourceNameType
    , @Parm37           WBSourceNameType
    , @Parm38           WBSourceNameType
    , @Parm39           WBSourceNameType
    , @Parm40           WBSourceNameType
    , @Parm41           WBSourceNameType
    , @Parm42           WBSourceNameType
```

```
, @Parm43          WBSourceNameType
, @Parm44          WBSourceNameType
, @Parm45          WBSourceNameType
, @Parm46          WBSourceNameType
, @Parm47          WBSourceNameType
, @Parm48          WBSourceNameType
, @Parm49          WBSourceNameType
, @Parm50          WBSourceNameType
) AS
DECLARE
  @RowPointer RowPointer
, @Whse       WhseType
, @TmpAmount  AmountType
, @StartWhse  WhseType
, @EndWhse    WhseType
, @StartItem  ItemType
, @EndItem    ItemType
, @StartLoc   LocType
, @EndLoc     LocType
DECLARE @ttItemloc TABLE (
  RowPointer uniqueidentifier
, whse       nvarchar(4)
, item       nvarchar(30)
, loc        nvarchar(15)
, amount     decimal(20,8)
, processed  tinyint
)
SET @StartWhse = ISNULL(@Parm1, dbo.LowString('WhseType'))
SET @EndWhse   = ISNULL(@Parm1, dbo.HighString('WhseType'))
SET @StartItem = ISNULL(@Parm2, dbo.LowString('ItemType'))
SET @EndItem   = ISNULL(@Parm2, dbo.HighString('ItemType'))
SET @StartLoc  = ISNULL(@Parm3, dbo.LowString('LocType'))
SET @EndLoc    = ISNULL(@Parm3, dbo.HighString('LocType'))
INSERT INTO @ttItemloc
SELECT RowPointer, whse, item, loc, 0, 0
```

```
FROM itemloc
WHERE whse BETWEEN @StartWhse AND @EndWhse
  AND item BETWEEN @StartItem AND @EndItem
  AND loc  BETWEEN @StartLoc  AND @EndLoc
WHILE EXISTS (SELECT TOP 1 1 FROM @ttItemloc WHERE processed = 0)
BEGIN
   SELECT TOP 1 @RowPointer = RowPointer
   FROM @TTItemloc
   WHERE processed = 0
   SET @TmpAmount = 0
   EXEC SSSWBCanInvValSubItemlocSp @RowPointer, @TmpAmount OUTPUT
   UPDATE @ttItemloc
   SET amount = @TmpAmount
   , processed = 1
   WHERE RowPointer = @RowPointer
END
INSERT INTO #tt_drill_results(
  CHAR01, CHAR02, CHAR03, DECI01, amount, RowPointer
)
SELECT whse, item, loc, amount, amount, RowPointer
FROM @ttItemloc
RETURN 0
```

DataViews

About DataViews

With DataViews, you have the ability to view, parse, sort, group by, and filter data from the application within user-definable views and layouts. The DataView setup is similar to an Excel spreadsheet, displaying data in columns. As with form personalization, the DataView structure supports user, group, and site level layouts. If needed, you can export the DataView to a printer, XPS file, or Excel .xls file. Additionally, you can navigate to the related maintenance form by clicking **Details** on the right-click context menu.

Types of DataViews

These types of DataViews exist:

- **Predefined**: This DataView is constructed through the DataViews setup process and is available to users who have access.
- **Form**: You can generate this DataView by clicking the DataView button in the toolbar. Data populates from the active form that you are viewing.
- **Drilldown**: This DataView is associated with a critical number and includes records related to the critical number calculation.

DataView Setup

Use these forms to set up a DataView:

- DataViews Setup
- DataView IDO Setup

You can create a repository of DataViews for specific users and groups. To create a new DataView, you must select a base IDO. All properties of the IDO selected are then listed to allow the user to choose the exact information that they want to include in the DataView. The ability to customize the display name of a property is supported.

Advanced functionality, such as IDO linking and sorting options, can be populated on the **DataView IDO Setup** form.

DataView Results

A DataView Generation button is available for all forms in the application. When you click the button, the data displayed on the current form, obeying applied filters, is sent to the **DataView Results** form. On this form, you can group, sort, and sum data to promote advanced data analysis. To generate a

hardcopy of the data, you can save the results as an Excel or PDF file, or send them to a network printer.

Saved Layouts

Each time DataView results are presented, you can save a copy of the layout, which includes any personalizations you have made. This eliminates the redundant task of grouping and sorting the results. Depending on how user permissions are configured, some users can save their changes to the different scope levels: user, group, or site.

Right-Click Actions Menu

On some DataViews, you can right-click on a column and select various actions. For example, you can drill down into the form where data maintenance or transaction entry is performed. You can also drill down into another DataView, or run an executable program, using the **Actions** menu.

The actions that are available on any DataView column are determined on the **DataView Actions** form. For more information, see Setting Up the Right-Click Action Menu for DataViews.

Security

Provisions can be put in place to control which data elements are available to which users. You can do this two ways:

- Filter the IDOs to be displayed during DataView creation.
- Specify user permissions for form DataViews on the **DataViews Setup** form.

Multi-Row and Multi-Column Layouts

DataViews and DataSearch results can include multi-row (stacked) layouts.

About DataView Layouts

The layout of a DataView includes anything the user can change, including summaries, columns, groups, and filtering. You can save the data as you have arranged it, and recall this layout later. If you need to view the data in multiple ways, you can save multiple layouts from the same DataView. You can save layouts at the Vendor, Site, Group, and User scope levels.

Use these fields and buttons on the **DataView Layouts** form to describe a layout:

Source Type

This field shows the type of source:

- **Predefined**: This DataView was constructed through the DataViews Setup process, and is available to users who are given access.
- **Form**: You can generate this DataView by clicking the DataView button in the toolbar. Data populates from the active form that you are viewing.

- **Drilldown**: This DataView is associated with a critical number, and includes records related to the critical number calculation.

DataSearch Source: This is related to the DataSearch feature. Each source can have a different layout.

DataView

This field shows the name of the DataView.

Layout

This field shows a name for the layout that is used to distinguish it from other versions of the layout. On new records, specify the name for the layout to distinguish it from existing layouts.

Default

Select this check box to specify that this layout is used for the initial display of the DataView.

Scope Type

This field shows the level at which the layout is available: Vendor, Site, Group, or User.

Scope Name

For Scope Types Group and User, the specific group or user is displayed. On new records, select the specific group or user.

For the Scope Types Vendor and Type, this field is disabled.

Copying a Layout

Click **Copy Layout** to create a new layout based on the currently selected layout. The default name of the new layout comes from the previous layout.

Setting Up DataViews

Use the **DataViews Setup** form to maintain DataViews, select the IDOs to use when results are displayed to the user, designate user and group permissions, and select data layout options.

Setting Up New DataViews

To set up a new DataView:

1. On the **DataViews Setup** form, turn off Filter-in-Place.
2. Click in the DataView grid, then select **Actions > New** to create a new DataView.
3. In the **DataView** field, specify a name that best describes the data to be presented.
4. On the **General** tab, specify this information:
   - Select an IDO to include on the DataView.
   - Click **IDO Setup** to launch the **DataView IDO Setup** form, filtered for the currently selected IDO.

- Follow the steps in Setting Up an IDO for a DataView.
- The IDO grid summarizes the information populated on the **DataView IDO Setup** form. All IDOs and their relationships can be viewed within the grid.
- For editable DataViews, use the **Property Selection** grid to select the properties to be displayed in the **DataView Results** form:

The description for each property can clarify indiscernible property names.

The column label on the **Results** grid for each property shows the default caption.

Use the **Caption Override** column to specify a different column label in the Results grid.

5 Click **Save**.

NOTE: This application is prepackaged with multiple DataViews. Prepackaged DataViews cannot be directly modified, but they can be copied to a new DataView name and altered.

Creating a DataView from an Existing DataView

To create a DataView from an existing one:

1 Click Copy DataView to launch the DataViews Copy form

2 Copy the default values from the currently selected DataView.

3 Specify a new name for the DataView.

4 Click Save.

Setting Additional Information for the DataView

Use the tabs to set additional information:

1 On the **User Permissions** tab, specify this information:

- In the **Users** grid, grant specific permissions to existing users to view the DataView.
- In the **Groups** grid, grant specific permissions to existing groups to view the DataView.

NOTE: In order to see DataView results, the user must be listed here, or must be a member of a group listed here.

2 On the **Layouts** tab, specify this information:

- Click **Copy Layout** to duplicate the currently selected row. Use this functionality for managing end-user layouts. If you make a change in one department, you can deploy it to another. If you save a layout at the site level that should not have been saved, you can remove it.
- The layout grid shows the layouts available for the DataView.
- If the Scope Type is **Group** or **User**, the **Scope Name** field is enabled to select the specific group or user. If the Scope Type is **Vendor** or **Site**, **Scope Name** is disabled.

Other Actions

Click **Launch** to execute the DataView, and open the **DataView Results** form for the currently selected DataView.

Click **Import/Export** to launch the **Error! Hyperlink reference not valid.** form, filtered for the selected DataView.

Displaying a DataView

Use the **DataViews** form to view the stored DataViews and layouts, and launch the **DataView Results** form. The stored DataViews are preconfigured in the system. Use the results to drive *ad hoc* reporting.

To display a DataView:

1. Click **Filter-in-Place** in the toolbar to display the list of stored DataViews.
2. Select a DataView from the list.
3. If multiple layouts exist for the DataView, select a specific layout. Use the fields on this form to determine which layout to view:

   Layout

   This field provides a description to distinguish one version of the DataView layout from another.

   Scope Type

   This field specifies the level at which the DataView layout is available: site, group, or user.

   Scope Name

   This field specifies the group or user for which this layout is designed.

   Default

   This field is selected for the layout that takes precedence when more than one layout exists for the same scope type and name.
4. Click **Launch** to open the **DataView Results** form for the selected DataView layout.

Displaying DataView Results

DataView results can be displayed in three ways:

- You can launch the **DataView Results** form from the **DataViews** form, displaying only predefined DataViews.
- You can launch the **DataView Form Results** form from the **DataViews** form, displaying only form DataViews.
- You can create an event to launch the **DataView Results** form directly from a form you choose.

See Creating an Event to Call DataView Results for the steps to add an event.

Data

The **DataView Results** form and **DataView Form Results** form show data based on the properties that have been set on the **DataViews Setup** form and the **DataView IDO Setup** form. Properties differ by DataView and layout.

Options

You can specify this information on the **DataView Results** form:

Layout

If permissions are granted, you can save the current layout as the default layout or under a new layout name. You can delete and select existing layouts.

Display

Use these options to manipulate the data in the results grid:

- **Expand All**: Show all rows of data.
- **Collapse All**: Show only the top level rows of data.
- **Choose Columns**: Provide a full list of columns available to hide or display in the grid.
- **Custom Columns**: Launch a form to create new columns from existing columns in the data results.
- **Show/Hide Options**: Show or hide the Group By, Summary, Filter, and Column Pinning icons in the column header.
- **Show/Hide Group By**: Show or hide the Group By icon in the column header.
- **Show/Hide Column Scroll Lock**: Show or hide the Column Pinning icon in the column header.
- **Show/Hide Filters**: Show or hide the Filter icon in the column header.
- **Show/Hide Summaries**: Show or hide the Summary icon in the column header.
- **Show/Hide Extended Captions**: Show or hide the full caption including property name in the column header.

Print

This menu provides standard print options.

Send To

Use this menu to send the data to a XPS or Excel .xls file.

Refresh

When available, use this option to reload the results grid, as if you had closed and reopened it.

Setup

When available, use this option to launch the **DataViews Setup** form pre-filtered to the current DataView.

Notes

If you do not have permission to see the results, an authorization message is displayed instead of the results.

If you cannot view costs, and cost properties are included in the DataView layout, then no data is displayed. In this scenario, we recommend that you copy the DataView to a new name, and deselect the cost properties.

Setting Up a DataView Filter

To launch the DataView Filter Setup form, click Filter on the DataView IDO Setup form.

To build the filter:

1. Select the **Property Name** of the IDO to be evaluated.
2. Select the **Operation Method**.

   If you select **Is Null** or **Is Not Null**, the comparison field and value do not need to be specified.
3. Select the Comparison Type:
   - **Literal**: The property is compared to a hard-coded value.
   - **Property Name**: One property of the IDO is compared to another.
4. Specify a Comparison Value.
5. Click Add to translate the information from the fields into filter syntax, and show them in the editor box. Click Remove to erase the content in the editor box.

For complex comparison logic, it may be necessary to add parentheses around OR and AND statements, to make sure the filter is evaluated properly. To accomplish this task, manually edit the data in the editor box.

Because this is a modal form, clicking **Cancel** does not save changes. Click **OK** to save changes.

NOTE: To get a better understanding of how filters can be used, look at the vendor level DataViews provided as part of the application installation.

Setting Up the Right-Click Actions Menu for DataViews

Use the **DataView Actions** form to set up right-click menu actions for DataView columns. You can specify the forms to open, executables to run, and so on. The options you specify are displayed alphabetically in the dynamic menu list. You can specify a menu action at a class level so that the action is displayed each time a property of that class is displayed. For example, actions associated with the CustNum class are available on any DataViews that display a customer number field. You can also associate a menu action with a specific IDO and/or property to limit the action's availability to a more specific set of DataViews.

NOTE: The options you specify in the dynamic menu list are displayed alphabetically within Action Type.

Each time a user right-clicks on a cell in a DataView, the system dynamically builds the menu options based on the menu actions that are defined for the class, IDO and property of the underlying DataView value. The user can perform any of these types of actions that are defined:

- Launch a specific form, filtered to show values from the selected DataView record.
- Run an executable (program) and pass it parameters that are values from the selected DataView record.
- Launch another, predefined DataView.
- Perform a global search, available on all columns, that launches the **DataSearch** form, filtered with values from the selected DataView record.

Setting the Caption for the Menu Option

In the **Caption** field, specify the text that displays for this action in the right-click menu. You can specify a string name here if you want the option to be translatable.

This field can be used to suppress multiple occurrences of the same command if the same action is available at different scope levels. Only one occurrence of a caption with the same name is displayed. If there are duplicate actions, the more specific level takes precedence: User, then Group, then Site, then Vendor.

Displaying the Action on the Action Menu

If the action should be displayed on the **Action** menu, select **Active**. Clear this field to temporarily disable an action.

Specifying "Applies To" Information

To set up actions that are shared by multiple DataViews and columns, use the **Applies To** section of the **DataView Actions** form to create a structure that identifies the cases where an action is displayed. Specify which property class, IDO, or property should have access to the action you are defining. If any fields in this section are left blank, the action applies to everything in that group. For example, if you specify a Class Name of **CustNum**, the menu action is enabled for every DataView column that is associated with a customer number. However, if you specify the IDO as **WDFSCustomers**, then only those DataViews that are built using the WBFSCustomers IDO display the menu action.

Specify this information:

1. Specify the Scope level to which this action applies: Vendor, Site, Group, or User. If the scope is Group or User, specify the group or user name to which it applies.
2. Specify the Class Name of an IDO property class to which this action applies.

Specifying "Action" Information

Specify this information to identify the action to take when a user selects the menu option:

1 Specify the Action Type to be performed by this menu option: Run DataView, Run Form, or Run Executable.

- If the Action Type is **Run DataView**, specify the name of the DataView, the name of the layout to use when running the DataView, and the Filter Property, which is described below in "About Filters."
- If the Action Type is **Run Form**, specify the Form Name of the form to open and the Initial Command (Refresh, Add, FilterInPlace) to run on the target form when the action is called. Specify the Filter Property, which is described below in "About Filters."

  Specify any variables to set on the target form. This must be a comma-separated list of variables and the values to which to set them. For example, for a DataView action that opens the Order Verification Report using the selected CoNum in the DataView, set the value in SetVariables to **OrderStarting=FP(CoNum),OrderEnding=FP(CoNum)** where FP is a substitution keyword. This example sets both the starting and ending customer order number range of the report to the value of the CoNum in the DataView where you selected the action. The substitution keyword CURPROP() could be used instead of CoNum if there is a possibility that the property might have a different name, for example, CoCoNum. For more information, see "Substitution Keywords" below.
- If the Action Type is **Run Executable**, click **Browse** and select the path and filename of the executable program that you want to run when the action is selected.

  Note: The executable runs on the client. If users selecting this action cannot access the executable file on their local computers using the path you specify here, an error message is displayed.

2 If the Action Type is **Run Form** or **Run Executable** you can apply extra filters in addition to the one in the **Filter Property** field. Click **Additional Filter**. The **DataView Actions Filter Setup** form is displayed. Substitutions are supported in this field, as described below in "Substitution Keywords." For more information, see DataView Actions Filter Setup.

About Filters

If the Action Type is **Run DataView** or **Run Form**, you can use the **Filter Property** and **Additional Filter** fields to specify how you want to filter the resulting DataView or form.

The **Filter Property** is the property on the target form or DataView to which the value of the current property will be filtered. For example, if you define an action with the caption "Item Details", where the Action Type is "Run Form" and the Form Name is Items, then if a user right-clicks on the item number **CP-10000** in a DataView and selects **Item Details**, the **Filter Property** is the property in the **Items** form that is filtered by CP-10000.

You can apply additional filters with the **Additional Filter** field and button.

Substitution Keywords

Substitutions are supported in the **Additional Filter**, **Command Line Parameters**, and **Set Variables** fields. Supported substitution keywords are P(...), FP(...), and CURPROP(). These keywords work the same way that they work in design mode, except that P and FP refer to the properties in the current row of the DataView instead of the form collection, and CURPROP() refers to the name of the property on which the user right-clicked. Substitutions can be used in these cases:

- To filter by additional values in the collection
- To set variables on target forms from values in the DataView
- To pass a value from the DataView to an executable in a command line parameter

Setting Up an IDO for a DataView

Use the **DataView IDO Setup** form when configuring an IDO for a DataView.

You must launch this form from the **DataViews Setup** form.

To set up the IDO on the **DataView IDO Setup** form:

1 In the IDO section, specify this information:

IDO

This field shows the currently selected IDO from the **DataView Setup** form.

Parent

This field shows a parent IDO if the IDO selected on the **DataView Setup** form has one.

Filter

Click this button to launch the **DataView Filter Setup** form to write filter syntax.

Record Cap

Select the number of records the DataView query returns in the results grid. For this menu, zero means all.

2 When a non-primary IDO is selected on the **DataViews Setup** form, the **Link By** section is enabled. Specify this information:

Link Type

Select multi-level, which nests the child IDO information under the parent, or single-level, which links them on the same row.

Parent Property

Select which property to use to join the parent to the child IDO.

Child Property

Select which property to use to join the child to the parent IDO.

3 In the **Order By** section, specify this information:

Property

Select a property of the IDO by which the output is ordered.

Order

Select ascending or descending, depending on if you want to order the values from lowest to highest (A to Z), or highest to lowest (Z to A).

NOTE: You can override the Order By clause in the DataSearch Results by using a Sort By statement at the DataView Layout level.

4 In the **Properties** section, specify this information:

Select All

Click this button to select all rows in the properties grid.

Deselect All

Click this button to deselect all rows in the properties grid.

Selected

Select the properties of the IDO that should be displayed in the DataView results.

5 Click **OK** to save the information or click **Cancel** to discard the information.

For both the **Link By** and **Order By** sections, click **Add** to create a new entry in the text box, and click **Remove** to erase everything in the text box.

DataSearch

Configuring Data Sources for DataSearch

You can use the **DataSearch Source Setup** form to configure which IDOs (sources) to search.

The **DataSearch** form has some built-in data sources. You can configure additional data sources through the **DataSearch Source Setup** form. Follow these steps to configure a new data source:

1 Specify the name and associated IDO for the source. Each DataSearch source can use only one IDO.

NOTE: To search a multi-level structure, such as Orders and Order Lines, you must either set up multiple DataSearch sources or add the appropriate columns from the second tier (Order Lines) to the selected IDO (Orders).

2 To filter the IDO to limit the search results, define the IDO filter expression.

For example, in the Customer Order Lines source, you could specify **Stat='O' AND QtyShipped < QtyOrdered** to only search unshipped/open lines. Click **Filter** to display a dialog box to help you set up the filter expression. For more information, see DataSearch Source Filter Setup.

3 Use the **Record Cap** fields to limit the amount of data that is initially displayed in the search results. Record Cap options are:

- **Use System Setting**: Use the Mongoose record cap setting.
- **Use Specified Max**: Specify a maximum value in the **Record Cap Value** field.
- **Retrieve All**: This option can affect system performance if many records are retrieved.

4 Optionally, specify a caption that displays in the Source column on the search results. The value you specify is translated if it is a string; otherwise, the literal value is displayed. You can use this field to define a translatable name for the search source. If this field is blank, the source name is used as the caption.

5 Use the Order By fields to sort the search results. Select a property and an order (ascending or descending) and click **Add** to create the OrderBy expression.

NOTE: You can override the Order By clause in the DataSearch Results by using a Sort By statement at the DataView Layout level.

6 On the General tab, select the IDO properties to be searched, as well as the IDO properties to be displayed in the results. Click **Select Search Properties** to select the same properties in the "To Show" grid that are selected in the "To Search" grid.

You can select additional information to show in the search results that you do not want to include in the search. For example, if you set up a search on customer name, you could include the customer address in the "To Show" results but not in the "To Search" list.

7 On the User Permissions tab, specify which users or groups can view results from searches performed using this DataSearch source.

8 Click **Launch** to launch the **DataSearch** form with this source selected.

9 Customize how the DataSearch results should be displayed. See the "Customizing and Saving DataView Layouts" section in Searching the System with DataSearch.

Searching the System with DataSearch

Use the **DataSearch** form to search for information stored anywhere in the application. The search results are listed numerically or alphabetically by data source. For example, you could search for all instances of the text string "Young" across all data sources. The results list every data source in the system where "Young" is found, for example in customers, orders, items, vendors, purchase orders, and so on. You can expand the data source to see a list of every occurrence within that data source.

Searching

To search for a value:

1 Open the **DataSearch** form. To open the form, use one of these options:

- Select the form from the Explorer.
- Click the DataSearch icon on the toolbar.
- Right-click in certain fields and select **DataSearch**.

2 Specify the search value, which can include an asterisk (*) as a wildcard.

3 Specify how to interpret the value: find any of these words, find all of these words, find words that contain this text, etc.

4 Optionally, specify which data sources to look in.

By default, all data sources are used.

NOTE: You can define additional custom data sources. See Configuring Data Sources for DataSearch.

5 Click the search icon to start the search. The results are displayed, initially grouped by data source. The **Count** field indicates how many records in the data source include the search term. Click the + button to expand any data source to view the list of results in a grid.

DataView Options

These standard DataView options are also available on this form:

- Click **Layout** and select **Save** to save the layout presentation (column order, show/hide columns, order by) of each DataSearch source for future use, or select **Delete** to remove the existing layout for the source.

- Click **Display** and select **Expand All** or **Collapse All** to expand or collapse all of the groups.
- Select **Choose Columns** if you want to rearrange, hide, or show columns using the Column Chooser.
- Click **Print** to print the results.
- Click **Send To** to export the results to Excel or XPS.

Customizing and Saving Layouts

To customize the display of the search results and save it for future use:

1. Change the columns of the search results by dragging and dropping columns, hiding or showing columns, and/or clicking the column headers to set the sort order.
2. When you like the layout of the results, click **Layout** and select **Save** to save the layout.
3. Set the **Scope Type** appropriately.
4. Save the layout record.

Notes

Each DataSearch Source has its own layout. For example, the Items DataSearch source can be grouped and sorted differently from the Customers DataSearch source.

Only one layout can be associated with each scope of a DataSearch, so users cannot have multiple, named layouts for a DataSearch source.

DataSearch source layouts do not support summaries, calculated columns, grouping or filtering.

DataSearch source layouts can also be viewed and maintained on the **DataView Layouts** form.

The search results (that is, the display of Source and Count) also have a layout, so you can choose how to sort the results and select the style to use when the results are displayed. This layout is saved with the other layouts for specific sources. The default vendor layout displays source by count, descending, so the source with the highest count is displayed first.

Form Synchronization

Understanding WinStudio Customizations

About WinStudio customizations

WinStudio customizations can affect all areas of the client tier:

- **Forms and components** - Authorized users can add, copy, or alter forms. They can also add, delete, recaption, hide, move, or otherwise alter components on a form.
- **Global objects** - Themes, image files, strings, variables, shortcut menus, validators, scripts, component classes, and property class extensions are all global objects that can be used by more than one form or component. Authorized users can add, delete, change, or associate global objects with different forms or components.
- **The Explorer window** - Authorized users can customize public folders. All users can customize the folders in **My Folders**. An application can also have customized versions of folders in the **Master Explorer**.

About customization versions

All customizations are stored in the forms database. Forms and global objects can be customized for one user, for a group of users, or for all users. As a result, the forms database can include more than one version of the same form or global object.

- The *Vendor* version of a form or global object is the version originally supplied by your vendor. It is unaffected by customizations.
- A *Site* version of a form or global object is a customized version that all users at a particular site can access, unless it is superseded by a *Group* or *User* version. Only a user with Site Developer editing permissions can create a Site version.
- A *Group* version of a form or global object is a customized version that all users in a particular group can access, unless it is superseded by a *User* version. Only a user with Site Developer editing permissions can create a Group version.
- A *User* version of a form or global object is a customized version that only one user can access (the user who created it). Users with Basic, Full User, or Site Developer editing permissions can create User versions.

About basic and major customizations

Basic WinStudio customizations are simple changes to forms and components that do not affect functionality. Users with Basic editing permissions can make only the following changes:

Components	Forms
Captions	Captions
Default Values	Dimensions (size)*
Dimensions (size, position)	Splitter settings*
Show or Hide	Grid column sequence*
Read Only/Disabled	Grid column width*
No Clear on New	Grid column visibility*
Default on Copy	
Upper or Lower Case	

NOTE: *These changes can be made by any end user, regardless of editing permissions, without the system having to save a full User-scope version of the form. If you have set your User Preferences to prompt you to save form, splitter, and grid changes, WinStudio prompts you when closing the form after making such changes. Otherwise, the changes are saved without prompting.

You should be aware, too, that system administrators can override the ability to save run-time form changes like this with the **Allow saving form runtime changes** process default.

Changes other than Basic changes can affect functionality and are referred to as *Major* changes. Developers with Full User or Site Developer editing permissions can make Basic changes as well as the more extensive, Major customizations.

The default settings for form synchronization using the **Form Sync** form:

- Automatically merge Basic form and component customizations with the new vendor version without prompting.
- Prompt you about how to handle Major customizations to forms and global objects. You can choose to:
  - **Keep** the customization without changing it.
  - **Remove** the customization, replacing it with the new vendor version.

About Synchronization

WinStudio serves as a development tool for customizing forms. "Form Sync" is an aid to preserving customizations when you apply an upgrade to an application. The process of handling customizations in an upgrade is referred to as *synchronization*.

CAUTION: Effective synchronization demands a thorough understanding of the process involved, the preparation required, and the level and type of synchronization desired. We strongly recommend that you take time to familiarize yourself with these concepts and practices before you perform any synchronization tasks. For more information on these topics, see the **Related Topics** at the end of this topic.

Use the **Form Sync** form to manage, synchronize, and maintain customizations to forms in an application. You can use this form to selectively apply upgrades supplied by a software vendor and to manage customizations during the development process. Upgrades include vendor-supplied upgrades to an application, service packs, and single fixes. (A single fix applies to one form or a small set of forms and global objects.)

If your forms database contains customizations to objects affected by the upgrade, you should use Form Sync to synchronize your customized versions with the new versions. Synchronization allows you to preserve customizations when the upgraded application is installed, synchronizing your customizations with the changes introduced by the vendor.

NOTE: Synchronization applies only to customizations made with WinStudio, that is, customizations to forms and global objects in the client tier. It does not apply to modifications to IDOs, stored procedures, triggers, or other areas of an application. For more information, see Understanding WinStudio Customizations.

Synchronization replaces older versions in the current forms database with newer versions. During this process, you decide how to handle customizations (User, Group, and Site versions) of forms and global objects. You can retain a customization, replace it with an updated base-level version, or, in some cases, edit it.

Synchronization requires two configurations, a *Source* configuration and a *Target* configuration. Synchronization takes place using the forms databases in the two configurations. For more information, see About Source and Target Configuration Selection.

There is a variety of ways you can perform synchronization. You can choose to synchronize:

- Forms only
- Global objects only
- The Explorer only
- Any combination of these

You can also choose to keep or replace customized versions of forms, global objects, and/or the Explorer.

After you perform your synchronization, you should always test the results of your efforts before deploying the resulting synchronized files to the production environment. For more information, see Testing Synchronization Results.

Before You Use Form Sync

Before you use Form Sync, take these actions to prepare your system:

- Back up your databases.

- Verify that your database servers are configured properly.
- Determine what level of customization synchronization you need to perform.
- Install any third-party products or databases you want to synchronize.
- (Optional) Select the Source and Target configurations you want to use.

  This action is necessary only if the configurations you want to use were not selected when you first opened Form Sync.

About Source and Target Configuration Selection

Synchronization requires two configurations, a *Source* configuration and a *Target* configuration. Synchronization takes place using the forms databases in the two configurations:

- The **Source** forms database contains upgrades from the vendor. During synchronization, the upgrades are copied from the Source to the Target forms database. The Source database is used only during synchronization and does not figure subsequently in a production or development environment. Typically, the Source database is a new forms database supplied by the vendor or a copy of your current forms database to which you have applied the vendor's upgrade.
- The **Target** forms database is a *copy* of your current forms database that contains your customizations. During synchronization, the Target database receives upgrades from the Source database. The end result of the synchronization process is a Target forms database that contains upgrades merged with your customizations.

To create a Target database, first back up your current forms database and then restore a copy of it. After synchronization, this Target database serves as your new production forms database.

For details on creating Source and Target configurations, see the upgrade documentation from your vendor.

Select Source and Target configurations according to the type of task you want to perform. Use the following tables to help you know how to select the proper configurations.

NOTE: In a configuration, **local** is an invalid server name. You must use the name of the SQL Server instance on the local machine.

Synchronization Tasks

To do this...	For the Source, select...	For the Target, select...
Synchronize custom objects with new Vendor versions of the objects.	A configuration that includes a forms database and application database that contain new Vendor versions of objects.	A configuration that includes a forms database and an application database that contain your customized objects. Customized objects

		can be of the scope types Site, Group, or User.
Synchronize custom objects with other custom objects that serve as base versions.	A configuration that includes a forms database containing custom objects that you want to use as base versions.	A configuration that includes a forms database containing custom objects that you want to synchronize with the base versions.

NOTE: If the Source and Target forms databases reside on different servers, the server that contains the Source forms database must be defined as a linked server in the server that contains the Target forms database. For instructions to configure linked servers, see the Microsoft SQL Server documentation.

About Default Synchronization

We recommend the default settings for synchronization because many customizations are merged automatically, which reduces editing time afterwards. This topic describes the default synchronization process.

Forms

In analyzing a customized form, Form Sync makes a three-way comparison between these three versions:

- The customization in the Target configuration
- The base-level version in the Target configuration
- The base-level version in the Source configuration

During analysis, Form Sync looks at component attributes, such as a component's data source or type, and form attributes, such as a form's caption or script.

If the base-level version of an attribute in the Source is:

- The same as the base-level version in the Target, Form Sync keeps the customized version of the attribute in the Target without prompting the user.
- Different from the base-level version in the Target, Form Sync does this:
  - For customizations to Basic attributes, Form Sync keeps the customization in the Target without prompting.
  - For customizations to Major attributes, Form Sync prompts you about how to handle the customization. You can keep it unchanged or replace it with the base-level version from the Source.

  NOTE: For descriptions of the differences between Basic and Major customizations, see Understanding WinStudio Customizations.

- Non-existent (that is, the form or component exists in the Target but *not* in the Source database) Form Sync leaves the new form or component in the Target database alone.

Form Sync then replaces base-level versions of forms in the Target with base-level versions from the Source.

Global Objects

A global object is treated as a unit in synchronization—unlike a form, whose multiple specifications for component attributes and form attributes are handled individually in the merge process. The only options for handling a customized global object are to:

- **Keep** the customized version.
- **Remove** the customized version, replacing it with the new version from the Source database.

If the base-level version of a global object in the Source is:

- The same as the base-level version in the Target, Form Sync keeps any customized versions of the object in the Target without prompting you.
- Different from the base-level version in the Target, Form Sync prompts you about how to handle the customization. You can either keep the customized object unchanged or replace it with the base-level version from the Source.
- Non-existent (that is, the global object exists in the Target but *not* in the Source database) Form Sync keeps the new global object in the Target database.

Form Sync then replaces base-level versions of global objects in the Target with base-level versions from the Source.

Explorer

When synchronizing the Explorer:

- Form Sync replaces the **Master Explorer** in the Target with the **Master Explorer** from the Source.
- Form Sync keeps all customized **Public** folders and **My Folders** folders in the Explorer in the Target database.

Because the **All Forms** folder is dynamically created from the forms database, the system automatically reflects any new base-level forms from the Source database.

About Synchronization with Site and Group Versions

You can use **Form Sync** to deploy revisions to a form customized at the **Site** or **Group** scope type so customizations for end users are synchronized with the revision. This approach simplifies the customization process, because end users do not have to revert their customized forms or recode them.

You can similarly deploy revisions to customized global objects.

Synchronization with site or group versions follows the same logic as synchronization with vendor versions in a vendor-supplied application upgrade. The Source configuration contains revised versions of forms customized at the **Site** or **Group** scope type. The Target configuration contains customized versions that are to receive the revisions. Note that developers revise forms in a separate development environment, which serves as the Source in synchronization.

To synchronize site or group versions, designate a base-level scope type in the **Synchronization base level** drop-down list in the **Synchronization Options** dialog box, and select either **Site** or **Group**. For more information, see the FormSync Help for this field.

Forms and global objects with the designated scope type serve as base-level versions for synchronization just as Vendor versions serve as base-level versions in an upgrade.

Definitions of forms with the designated scope type in the Source configuration are then merged with any customized forms that are based on forms with the same scope type in the Target configuration. Forms with the designated scope type in the Source configuration replace forms with the same scope type in the Target configuration.

EXAMPLE: A developer with Site Developer editing permissions adds a component to a form named **DueNow**. The developer specifies **Group** as scope type and **AccountsReceivable** as scope name.

A member of the **AccountsReceivable** group with the user ID **JJones** changes the caption for the component. The change is at **User** scope type, and the scope name is **JJones**.

In a separate development environment, the developer adds an event handler to the form.

In FormSync, the developer specifies **Group** as the synchronization base level, **DueNow** as the form name, the development configuration as the Source configuration, and the production configuration as the Target configuration. The developer then synchronizes the two configurations.

Following synchronization:

The **JJones** version of the form retains the customized caption *and* contains the new event handler.

The revised version of **DueNow** with scope type **Group** and scope name **AccountsReceivable** replaces the first customized version of the form with the same scope in the Target configuration.

The Vendor version of the form is unaffected.

About Messages and Prompts

About Form Sync Messages and Prompts

During the synchronization process, **Form Sync** generates messages and prompts. These messages and prompts can be recorded automatically in a log.

Messages

Messages are of three types:

- *Progress messages* report on the progress of the current process and are displayed in the status bar of the **Form Sync** form.
- *Warning messages* provide warnings about potential problems and are displayed in dialog boxes.
- *Error messages* present messages about errors that have taken place in the synchronization process and are displayed in dialog boxes.

You can set the level of detail for progress messages recorded in the log. The default level records main actions only and is recommended for most processes. Note that increasing the level of detail for these messages significantly increases processing time. For more information, see Log Detail Levels.

After processing is complete, you can read the messages in the log. For more information, see Viewing the Form Sync Log.

Prompts

Prompts are questions about customizations that cannot be handled automatically during synchronization. You control which customizations generate prompts by setting synchronization options in Form Sync. You can choose to be prompted about:

- Whether to merge or remove a customized form
- Whether to merge or remove a customized global object
- Whether to keep or remove customized event handlers, global scripts, and form scripts

During synchronization, Form Sync keeps track of all prompts and warning messages through an initial analysis of the forms and objects being synchronized.

After the pass through, Form Sync presents these messages to you for your responses. At each of these points, you can decide whether to keep your customizations or discard them. You can also cancel the entire synchronization process at any point in this process, because no actual changes have yet been made.

After you respond to all prompts and warning/error messages, assuming you did not cancel the operation, Form Sync proceeds to make all changes based on your responses.

Synchronization and Method Call Messages

A customized method call in a form may have a different number of parameters and different types of parameters than the corresponding base-level method call. Form Sync checks for these differences, using a collection of method calls performed in the application. Form Sync determines whether the number of parameters matches and whether the type (input or output) of each individual parameter matches.

During synchronization, Form Sync reports any differences as warning messages.

After synchronization, you can reconcile differences by editing the form. All warning messages about method-call parameters are automatically recorded in the log, regardless of the log-detail setting. You can review the messages by querying **Warning** messages in the log.

Synchronization and Event Handler Prompts

NOTE: The following information refers to WinStudio form events and handlers, and *not* to application system events and handlers.

To examine event handlers, Form Sync collects all the handlers for a customized event in the Target configuration and all the handlers for the corresponding event in the Source configuration. It then compares them to determine whether the event and the entire set of handlers match.

Form Sync analyzes non-matching handlers and determines whether base-level handlers in the Source can be inserted into the customized version in the Target.

If there is no automatic way to merge the changes, Form Sync presents a prompt. You can select **Keep** to retain the customized event handlers or **Remove** to replace the customizations in the Target with base-level versions from the Source.

Synchronization and Script Prompts

Form Sync displays prompts when the base-level version in the Source configuration is different from the base-level version in the Target configuration. As with other synchronization prompts, you can select **Keep** to retain the customized script, or you can select **Remove** to remove the customized script and replace it with the base-level version in the Source.

Synchronizing Third-Party Products

If your application includes third-party add-in products, you must install the products in the Source configuration before you synchronize your forms databases. The Source configuration must include all components of your system. There are two reasons for this requirement:

- Synchronization deletes Vendor versions of forms, including Vendor versions of any third-party products, from the Target configuration. After this deletion, synchronization inserts Vendor versions from the Source into the Target.

  If the required third-party products are not installed in the Source configuration, Form Sync cannot replace the deleted Vendor versions in the Target.

- Customized versions of third-party products in the Target cannot be synchronized with modifications, if any, in the Vendor versions of the third-party products unless Vendor versions of the products exist in the Source.

  If your Target configuration contains customized versions of third-party products, and you do not install the products in the Source configuration before synchronization, you cannot reliably

reinstall the third-party product and synchronize your changes later. The Target no longer contains Vendor-level third-party logic.

When Form Sync compares an old Vendor object, a new Vendor object, and a customized object, Form Sync assumes that you added something of the same name as the Vendor and *not* that the two objects have the same origin. Instead of merging custom attributes, Form Sync forces you to choose one or the other.

Synchronization Tasks

The primary use of Form Sync is to synchronize, or merge, different versions of forms, global objects, and the Explorer from different sources. You can elect to synchronize:

- All forms, global objects, and the Explorer at one time
- Forms, global objects, and the Explorer singly or in combinations

These topics all describe specific aspects of the synchronization process:

- Synchronizing with the Default Settings (synchronizing all forms, global objects, and the Explorer at one time)
- Synchronizing Forms Only
- Synchronizing Global Objects Only
- Synchronizing the Explorer Only
- Replacing Base-Level Versions, Leaving Customizations Unchanged
- Replacing Base-Level Versions, Removing Customizations
- About Synchronization with Site and Group Versions
- Testing Synchronization Results

Synchronizing with the Default Settings

When you use the default settings to synchronize, the process affects all customizations to all forms, all global objects, and the Explorer in the Target configuration. For a description of the process and its results, see About Default Synchronization.

CAUTION: Before synchronization, be careful to back up the forms database in the Target configuration and confirm your database and server settings. For more information, see Before You Use FormSync.

To synchronize with the default settings:

1. With the **Form Sync** form open to the main page, click the **Restore Defaults** button to ensure that all options are set to their defaults.
2. From the **Synchronization base level** drop-down list, specify whether you want to synchronize objects at the **Vendor**, **Site**, or **Group** level. For more information, see Synchronization base level.

3 Verify that the **Only process objects changed at the synchronization base level** and the **Process scripts as single objects** options are both selected.

4 From the **Log detail** drop-down list, select the level of detail you want to record in the log. For more information, see Log Detail Levels.

5 On the **Forms** tab, verify that the **Merge customized forms with base-level forms** option is selected.

6 Click **Synchronize**.

Form Sync proceeds with the synchronization process, using the default settings.

Synchronizing Forms Only

You can synchronize only the forms, without affecting global objects or the Explorer in the Target configuration.

When you use the form-only option, FormSync does this:

- Merges base-level versions of forms from the Source with customizations in the Target.
- Keeps customized global objects in the Target configuration without changing them.
- Keeps base-level versions of global objects in the Target configuration and does not replaced them with base-level versions from the Source configuration.
- Keeps the **Public** folders and **My Folders** folders in the Explorer of the Target configuration without changing them.

CAUTION: Before synchronization, make sure you back up the forms database in the Target configuration and confirm your database and server settings. For more information, see Before You Use FormSync.

To synchronize forms only:

1 Open the **Form Sync** form, specify your Source and Targe configurations, and click **OK**.

2 From the **Synchronization base level** drop-down list, specify whether you want to synchronize objects at the **Vendor**, **Site**, or **Group** level. For more information, see Synchronization base level.

3 In the **Log detail** field, specify the level of detail you want to record in the log. For more information, see Log Detail Levels.

4 Select the check box labeled **Only process objects changed at the synchronization base level**.

5 On the **Forms** tab, select **Merge customized forms with base-level forms**.

6 Click **Merge Options**.

7 On the **Merge Options** page, do one of these actions:

8 To use the default settings, click **Restore Defaults**.

9 Set options as desired on the **Matching Components** tab, the **Non-matching Components** tab, and the **Form Attributes** tab.

10 Click **OK**.

11 (Optional) In the **Form name** field, specify one or more forms to synchronize. For more information, see Form name.

12 (Optional) To remove all users' form run-time changes from the form, select the **Remove runtime changes when form replaced** option. For more information, see Understanding WinStudio Customizations.

13 On the **Explorer** tab, clear the **Update** check box, and select the **Keep customizations** check box.

14 On the **Global Objects** tab, select **Keep customizations**, and clear all check boxes in the **Filter** group box.

   To clear all the check boxes at once, use the **Toggle All/None** option at the end of the list.

15 Click Synchronize.

Synchronizing Global Objects Only

You can synchronize only global objects, without affecting forms or the Explorer in the Target configuration.

When you use the global objects-only procedure, Form Sync does this:

- Selectively replaces customized global objects in the Target configuration with base-level versions from the Source configuration.
- Deletes the old base-level versions of global objects in the Target configuration and loads the new base-level versions from the Source.
- Keeps customized forms in the Target configuration without changing them.
- Keeps base-level versions of forms in the Target configuration and does not replace them with base-level versions from the Source.
- Keeps **Public** folders and **My Folders** folders in the Explorer of the Target configuration unchanged. Changes to the Explorer in the Source configuration are not copied to the Target configuration.

CAUTION: Before synchronization, make sure you back up the forms database in the Target configuration and confirm your database and server settings. For more information, see Before You Use FormSync.

To synchronize global objects only:

1 Open the **Form Sync** form, specify your Source and Targe configurations, and click **OK**.

2 From the **Log detail** drop-down list, select the level of detail you want to record in the log. For more information, see Log Detail Levels.

3 From the **Synchronization base level** drop-down list, specify whether you want to synchronize objects at the **Vendor**, **Site**, or **Group** level. For more information, see Synchronization base level.

4 Select the check box labeled **Only process objects changed at the synchronization base level**.

5 On the **Forms** tab, select **Do not process forms**.

6 On the **Explorer** tab, clear the **Update** check box, and select the **Keep customizations** check box.

7 On the **Global Objects** tab, select both **Remove customizations** and **Prompt before removal**.

8 Specify which global objects to synchronize:

- Select the check box for each type of object that you want to synchronize, and clear those you do not want to synchronize.
- (Optional) Specify a filter for each specified type.

For more information about selecting and filtering global objects, see Filtering Global Objects During Synchronization.

9 Click Synchronize.

Synchronizing the Explorer Only

You can synchronize only the Explorer, without affecting either forms or global objects in the Target configuration.

When you synchronize the Explorer only, Form Sync does this:

- Replaces application folders in the **Master Explorer** of the Target configuration with folders from the Source configuration
- Keeps **Public** folders and **My Folders** folders in the Explorer of the Target configuration
- Keeps customized forms in the Target configuration
- Keeps base-level versions of forms in the Target configuration and does not replace them with base-level versions from the Source configuration
- Keeps customized global objects in the Target configuration
- Keeps base-level versions of global objects in the Target configuration and does not replace them with base-level versions from the Source configuration

CAUTION: Before synchronizing, make sure you back up the forms database in the Target configuration and confirm your database and server settings. For more information, see Before You Use FormSync.

To synchronize the Explorer only:

1 Open the **Form Sync** form, specify your Source and Targe configurations, and click **OK**.

2 In the main FormSync window, from the **Log detail** drop-down list, select the level of detail you want to record in the log. For more information, see Log Detail Levels.

3 From the **Synchronization base level** drop-down list, specify whether you want to synchronize objects at the **Vendor**, **Site**, or **Group** level. For more information, see Synchronization base level.

4 Select the check box labeled Only process objects changed at the synchronization base level.

5 On the Forms tab, select Do not process forms.

6 On the Explorer tab, select both the Update and Keep customizations check boxes.

7 On the Global Objects tab, select Keep customizations, and clear the check box next to each type of global object.

   To clear all the check boxes at once, use the **Toggle All/None** option at the end of the list.

8 Click Synchronize.

Replacing Base-Level Versions, Leaving Customizations Unchanged

You can replace uncustomized base-level versions in the Target configuration with base-level versions from the Source configuration, while leaving customized versions in the Target configuration unchanged.

> **CAUTION:** This procedure is generally not recommended because the customized versions are kept, preventing the new base-level versions from the Source configuration from being installed. This means that any enhancements and fixes in the new base-level versions are not available to those using customized forms until you manually merge the new base-level versions in the Target configuration with the customized versions.

When you use this option, Form Sync does this:

- Keeps customized versions of forms in the Target configuration
- Replaces uncustomized base-level versions of forms in the Target configuration with base-level versions from the Source configuration
- Keeps customized global objects in the Target configuration
- Replaces uncustomized base-level versions of global objects in the Target configuration with base-level versions from the Source configuration
- Keeps **Public** folders and **My Folders** folders in the Explorer of the Target configuration
- Replaces application folders in the **Master Explorer** of the Target configuration with folders from the Source configuration

> **CAUTION:** Before you synchronize, make sure you back up the forms database in the Target configuration and confirm your database and server settings. For more information, see Before You Use FormSync.

To replace only customized base-level versions, leaving customized versions unchanged:

1 Open the **Form Sync** form, specify your Source and Targe configurations, and click **OK**.

2 In the **Log detail** field, specify the level of detail you want to record in the log. For more information, see Log Detail Levels.

3 From the **Synchronization base level** drop-down list, specify whether you want to synchronize objects at the **Vendor**, **Site**, or **Group** level. For more information, see Synchronization base level.

4 Select the check box labeled **Only process objects changed at the synchronization base level**.

5 On the Forms tab, select Leave customized forms untouched and verify that the Remove runtime changes when form replaced option is cleared.

6 On the Explorer tab, select both the Update and the Keep customizations check boxes.

7 On the Global Objects tab, select Keep customizations and select the check box next to each type of global object.

8 Click Synchronize.

Replacing Base-Level Versions, Removing Customizations

You can replace base-level versions in the Target configuration with base-level versions from the Source configuration, and at the same time remove customized versions from the Target.

When you use this option, Form Sync does this:

- Removes customized forms from the Target configuration
- Replaces base-level versions of forms in the Target configuration with base-level versions from the Source configuration
- Removes customized global objects from the Target configuration
- Replaces base-level versions of global objects in the Target configuration with base-level versions from the Source configuration
- Removes **Public** folders and **My Folders** folders in the Explorer of the Target configuration
- Replaces application folders in the **Master Explorer** of the Target configuration with folders from the Source configuration

CAUTION: Before you synchronize, make sure you back up the forms database in the Target configuration and confirm your database and server settings. For more information, see Before You Use FormSync.

To replace base-level versions, at the same time removing customizations:

1 Open the **Form Sync** form, specify your Source and Targe configurations, and click **OK**.

2 From the **Synchronization base level** drop-down list, specify whether you want to synchronize objects at the **Vendor**, **Site**, or **Group** level. For more information, see Synchronization base level.

3 From the **Log detail** drop-down list, select the level of detail you want to record in the log. For more information, see Log Detail Levels.

4 On the **Forms** tab, select **Remove customized forms**.

CAUTION: Customizations that are kept override new base-level versions from the Source configuration. Enhancements and fixes in the new versions are not available to users of customized forms until you manually merge new versions in the Target with customizations.

5 To have Form Sync prompt you to remove or retain each customized form, select **Prompt before removal**.

6 (Optional) Set form **Filter** options to apply the synchronization only to specified forms.

7 To remove any run-time form changes that users have made, select **Remove runtime changes when form replaced**.

8 On the **Explorer** tab, select the **Update** check box and clear the **Keep customizations** check box.

9 On the **Global Objects** tab, select **Remove customizations**.

10 To have Form Sync prompt you to remove or retain each customized global object, select **Prompt before removal**.

> **CAUTION:** Customizations that are kept override new base-level versions from the Source configuration. Enhancements and fixes in the new versions of global objects are not available to users until you manually merge new versions in the Target with customizations.

11 Click Synchronize.

Testing Synchronization Results

After synchronizing versions of an application, you should examine and test customizations in the Target configuration.

A test environment must be parallel to the updated production environment, with forms accessing the updated IDOs, application database, and other parts of the application.

You should plan to test:

- Customized forms in which you kept a customization in response to a FormSync prompt
- New forms that you created, including copies of Vendor versions of forms saved under a new name
- Forms that use a customized global object that you kept in response to a FormSync prompt

Consider dependencies within a form and within the system. An upgrade can modify:

- An IDO or other resource on which the customized object is dependent
- A Vendor version of a form or global object on which the customized object is dependent
- A Vendor version of a form or global object that depends on the Vendor version of the customized object

The effects of such changes are not transparent, and they must be considered in your test plan.

When there is a change in component type (for example, from radio button to toolbar button), you might have to adjust the size and position of the component.

Form Sync log

During the synchronization process, Form Sync generates various messages and prompts. These messages and prompts can be recorded automatically in a log.

Viewing the Form Sync Log

During the synchronization process, depending on your settings, Form Sync can record progress and other messages in a log. You can later review this log to view Form Sync progress and any errors encountered during synchronization.

For more information about setting the level of detail of messages recorded in the log, see Log Detail Levels.

To display Form Sync log messages:

1 On the main page of the **Form Sync** form, click the **View Log** button.

By default, the log displays all recorded messages.

2 To retrieve only certain types of messages, select the check boxes next to the types you want to view. To omit messages of a particular type, clear the check box. Options include:

- Progress
- Warning
- Error

3 (Optional) In the **Text** field, specify a filter on message text:

- To retrieve all messages, leave this field blank.
- To retrieve messages that consist of a known string, type the string.
- To retrieve messages that contain a string, type part of the string and use the percent sign (%) as a wildcard character. For example, to retrieve only those messages that contain **abc**, type **%abc%**.

4 Click **Refresh**.

Log Detail Levels

You can set the level of detail for synchronization messages recorded in the Form Sync log:

- **None** – Records only warning and error messages.
- **Main Actions** – Records only high-level actions. This level reduces processing time compared with the **Detailed** setting.
- **Detailed** – Records all the individual steps Form Sync takes during processing. This setting significantly increases processing time.

The **Main Actions** and **Detailed** settings record messages of these types:

Main Actions

- Begin Copy
- Copy of <*n*> customized objects successfully completed
- Processing form <*scope_type*> <*scope_name*>
- Form successfully copied
- Form Licensing not copied. Copy Form Licensing option not selected.
- Form License ModuleMembers successfully copied
- No Form License ModuleMembers copied. Source Form was not a 'New' or 'Copied' form.
- Menu successfully copied
- All form variables replaced by new vendor version form variables
- All event handlers replaced by new vendor version event handlers
- Beginning synchronization
- Removing old vendor versions from target
- Copying new vendor versions from source to target
- Synchronization successfully completed
- Leaving custom-created form <*scope_name*>
- Beginning logged prompt replay
- Prompt replay successfully completed
- Saved customized form <*scope_name*>
- Removed customized form <*scope_name*>

Detailed

- User responded <*answer*> to prompt <*prompt*>
- Processing variable <scope_type> <*scope_name*>
- Variable successfully copied
- Processing string <*scope_type*> <*scope_name*> in table <*table_name*>
- String successfully copied
- Processing validator <*scope_type*> <*scope_name*>
- Validator successfully copied
- Processing script <*scope_type*> <*scope_name*>
- Script successfully copied
- Processing <*property_name*> <*scope_type*> <*scope_name*>
- <*property_name*> successfully copied
- Customized value for form miscellaneous attributes retained
- Customized value for form miscellaneous attributes replaced with new vendor value
- Form miscellaneous attributes replaced by new vendor version
- Customized value for form variable retained: <*variable_value*>
- Customized value for form variable replaced with new vendor value: <*variable_value*>

- Custom-created form variable retained: <*variable_value*>
- Custom-created form variable discarded: <*variable_value*>
- Customized attributes for event handler retained: <*event_handler_attributes*>
- Customized attributes for event handler replaced with new vendor values: <*event_handler_attributes*>
- Custom-created event handler retained: <*event_handler*>
- Custom-created event handler discarded: <*event_handler*>
- Processing component <*component_name*>
- Custom-created component retained: <*component_name*>
- Custom-created component discarded: <*component_name*>
- Processing ActiveX Script Subroutine <*activex_subroutine*>
- Customized version of ActiveX script retained
- Customized version of ActiveX subroutine replaced with new vendor version
- Customized value for <*field_description*> retained
- Customized value for <*field_description*> replaced with new vendor value
- Processing string <*scope_type*> <*scope_name*>
- leaving customized string <*scope_name*> because vendor version is unchanged
- deleted customized string <*scope_name*>
- leaving customized variable <*scope_name*> because vendor version is unchanged
- deleted customized variable <*scope_name*>
- leaving customized validator <*scope_name*> because vendor version is unchanged
- deleted customized validator <*scope_name*>
- Processing menu <scope_type> <*scope_name*>
- leaving customized menu <*scope_name*> because vendor version is unchanged
- deleted customized menu <*scope_name*>
- leaving customized script <*scope_name*> because vendor version is unchanged
- deleted customized script <*scope_name*>
- leaving customized <*property_name*> <*scope_name*> because vendor version is unchanged
- deleted customized <*property_name*> <*scope_name*>
- Loaded customized variable <*scope_name*>
- Loaded customized validator <*scope_name*>
- Loaded customized script <*scope_name*>
- Loaded customized string <*scope_name*> from table <*table_name*>
- Loaded customized menu <*scope_name*>
- Loaded customized property default <*scope_name*>
- Loaded customized form <*scope_name*>
- Loaded component <*component_name*>
- Loaded event handler <*event_handler_name*>
- Loaded form variable <*variable_name*>

- Removed customized form component <*component_name*>
- Removed customized event handler <*event_handler_name*>
- Removed customized form variable <*variable_name*>
- Removed customized variable <*scope_name*>
- Removed customized validator <*scope_name*>
- Removed customized script <*scope_name*>
- Removed customized string <*scope_name*>
- Removed customized menu <*scope_name*>
- Removed customized property default <*scope_name*>

Printing, Sorting, Searching, or Archiving Messages

You can copy messages from the log to the Windows clipboard and paste them into a spreadsheet. You can then use the spreadsheet to print, sort, search, or save the messages.

To copy messages from the log to a spreadsheet:

1. On the main page of the **Form Sync** form, click the **View Log** button.
2. Set the filter criteria for your messages.
3. Click **Refresh**.
4. Select the messages in the grid that you want to process.
5. Press Ctrl+C.
6. Open a spreadsheet application.
7. Paste the contents of the clipboard into a spreadsheet document.

Clearing the Form Sync Log

The clearing process deletes all records stored in the log.

To clear all messages and prompts from the log:

1. On the main page of the **Form Sync** form, click the **View Log** button.
2. In the **Form Sync Log** form, click **Clear**.
3. When prompted for confirmation, click **Yes**.

Index

A

B

C

D

E

F

I

J

M

O

P

R

S

T

U

V