Infor Mongoose Core Development Guide

Release 10.x

Publication Information

Release: Infor Cloudsuite Business (Industrial) 10.x
Publication Date: January 16, 2019
Document code: csbi_10.x_mgsdug_op_csind_en-us

Contents

Contacting Infor

If you have questions about Infor products, go to Infor Concierge at https://concierge.infor.com/ and create a support incident.

If we update this document after the product release, we will post the new version on the Infor Support Portal. To access documentation, select **Search > Browse Documentation**. We recommend that you check this portal periodically for updated documentation.

If you have comments about Infor documentation, contact documentation@infor.com.

Chapter 1: 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 application 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 extension class 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.

Constructing Application Messages

A message can be constructed so 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, for example &5 and &C.

These are possible sources of values for substitution parameters:

- 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 you construct 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.

For 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 these parameters:

- &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 looks 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 example might evaluate to this 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].

Setting Application 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 or **General 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 more information see Constructing Application Messages on page 9.

   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.
4. Save the message.

Invoking and Concatenating Multiple Application 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.

For example, consider these 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].

About Constraint Exception Messages

Describe the process or concept here

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.

Building Constraint Exception Messages

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**.

Chapter 2: 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.

IDO Elements

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.

Defininig an IDO

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.

The **IDOs** form is where you start to define an IDO. IDO definitions include:

- Attributes for the IDO itself
- References to tables in the application database
- Property definitions for table columns
- Metadata about methods defined for the IDO

You can access all of these elements from this form. You can also access the row-level security that can be defined for IDOs.

The metadata that you edit in the **IDOs** form is stored in the Objects database.

Existing IDOs must be checked out before they can be edited. You can only check out IDOs that have the same **Access As** setting as your environment.

For more information, see Access As Field.

For more information about the tasks you can perform, see these topics:

- Adding an IDO on page 27
- Viewing and Understanding an IDO Definition on page 17
- Editing IDOs on page 28
- Deleting an IDO on page 29
  Note: You can use deletion rules to determine what is to happen when one or more records meet specified criteria for other IDOs that reference properties in the one you want to delete. Define these deletion rules on the **Rules** tab. For more information, see IDO Deletion Rules on page 29.
- Editing or Removing IDO Tables on page 34
- Adding Base or Secondary Tables to IDOs on page 33
- Editing an IDO Property on page 39
- Adding a Property to an IDO on page 39
- Editing Methods on page 37
- Adding a Method to an IDO on page 36

Using an IDO

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**. See Adding an IDO on page 27.

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.

1 Open the **IDO Projects** form.

2 Select **Actions>New** or click the **Create a New Object** tool bar button.

3 Specify the name of your project. The name must follow these rules:

- 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 tool bar.

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**.

3 Save your changes.

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.
3 Use the fields in the **Attributes** group to view this identifying information about the IDO:
   - 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, see the context-sensitive help for the specific attribute or element.
4 In the source control status fields, view the revision status and check-out status of the IDO.
5 Use these buttons 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, as described in this table:

Button	Form that is displayed	Actions you can take on this form
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

Button	Form that is displayed	Actions you can take on this form
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

6 Use these tabs to view a summary of the elements and rules associated with the selected IDO:

Tab	Displays a list and condensed view of this information
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.

Multiple source control options are supported:

- 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 take these steps:

- 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.
- On the **Users** form, **Source Control** tab, configure the user profile for each system user who is authorized to check forms and other components in and out.

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

Comparing IDOs during check-in

Before you check in an IDO that you have modified, you can compare it with the current version. You can use a file comparison tool to compare the new and the current version of an IDO. You must set up a comparison tool in **User Preferences**.

When you click **Compare** on the **IDO Collections** form or the **Mass Check In** form, two XML files are created. The compare tool that you specified in **User Preferences** opens these two XML files to compare the contents.

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.

See Including Data from an External Database into a Mongoose Application on page 21.

Note: Currently, only SQL Server, Oracle, DB2, Postgres, and Progress data sources can be linked to Mongoose applications.

Unicode Support

Mongoose application 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 may need to set the “Non unicode literal” process default in the Mongoose application.

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** form. 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. The system maps whatever column you have assigned to use for optimistic locking to this column, as described above. During run time, Mongoose checks to see if this value has been modified elsewhere since the data was first queried, before your modifications.
- RowPointer, required to be a value that is unique for the entire table. For Outrigger data sources, this value is derived by concatenating the primary key values from each row in the external table.

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 Your 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.

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*
   Note: The database administrator must first create a linked server relationship using the SQL Server configuration tools.

   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 External Database into a Mongoose Application

Mongoose-based applications can communicate with external databases hosted on various types of servers through the IDO layer. To set up communication, use these steps.

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.

1. In the **Outrigger Profiles** form, create a profile for the database with these values:

   Note: This step is not necessary for a SQL Server database hosted on the same server as the Mongoose database, or for a SQL Server database hosted on a linked server.

   Database Type
   Specify the type of outrigger database to connect to.

   Data Source
   If you select `DB2` database type, specify the schema name.
   If you select `Oracle` database type with option Direct=False, specify the Oracle service name as defined in the *tnsnames*.ora file.

If you select `Oracle` database type with option Direct=True, specify the Oracle server's IP address or DNS name. See Notes below for additional options.
If you select **Postgres**, **Progress**, **Redshift**, or **SQL Server** as the database type, specify the database name.

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.

Profile Name
Specify the profile created in step 1.
For a SQL Server database hosted on the same server as the Mongoose database, or for a SQL Server database hosted on a linked server, leave this field blank.

Database Type
Defaults to the database type selected in Step 1, or `SQL Server` if the **Profile Name** is blank.

Database Name
For a SQL Server database hosted on the same server as the Mongoose database, specify the name of the database to which you want to link.
For a SQL Server database hosted on a linked server, you must also provide the name of the server using this format: databaseServer.databaseName.

Optimistic Lock Column Name
Optionally, specify the name of a column for optimistic locking that is present in some or all tables 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.

3 In the **Tables** grid, specify or select an existing table name in the external database.

Optionally, rename the **View Name** to avoid conflict with any tables or views in your Mongoose application or with other external tables being linked.

4 Save the table record to enable the buttons on the form.

5 Select the table and click **Columns**. This opens the **IDO Linked Tables** form, with an empty **Columns** grid.

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

The results can be adjusted by following these steps:

a Verify that the columns listed in the **Column Name** column match those of the external table.

b Optionally, rename the View Column Names as you want them to be shown in the **New IDO Wizard** form.

c Verify that the primary keys for the external table are correct, as specified in the **Keys** column. Add or modify the choices for keys as necessary.

Note: Keys are not initialized for Redshift databases, so you must specify them manually.

d Optionally, select the column to use for optimistic locking in the external table. Mongoose designates the NODATE literal string for the "RecordDate" value, and thus no optimistic locking is performed, when these conditions exist:

  - No "RecordDate" column exists in the external table.
  - The column specified in the **Optimistic Lock Column Name** field of the **IDO Linked Databases** form does not exist in the external table.
  - No column is designated for optimistic locking on the **IDO Linked Tables** form.

7. Click **Create View** to create a new view in the Mongoose database that defines the necessary columns and data types. The Mongoose view includes the columns from the external table, along with these columns and values:
   - "RecordDate", used for optimistic locking
   - "RowPointer", required to be a value that is unique for the entire table
8. Use the **New IDO Wizard** form to create an IDO whose Primary Base Table is the View Name that is created. The profile name is automatically populated, if applicable.
9. Use the **Advanced IDO Attributes** form to ensure that the proper Primary Keys are selected in the correct order.

Notes about the process:

- The profile that is 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 external 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 on page 24 for a code sample that uses this assembly.
- For DB2 databases, a third party driver package must be installed and configured. Contact Support for details.
- For Oracle and Progress databases, a license to a third party driver package must be purchased. Contact Support for details.
- There are two modes for connecting to Oracle databases:
  - If the application server is the Oracle server or Oracle Client software is installed on the application server, in the **Options grid**, specify an option of `Direct` with a value of `False`, and in **Data Source**, specify the Oracle service name (SID).
  - Otherwise, in the **Options grid**, add an option of `Direct` with a value of `True`, in **Data Source**, specify the Oracle server's IP address or DNS name, and add an additional option of `SID` with a value of the Oracle service name on the Oracle server. You can also add an option of `Port` with the port number to connect to. The default value is 1521. In this mode, you can perform write operations on a linked table in an Oracle database only through a configuration where the **Use Distributed Transactions (DTC)** flag is cleared.
- Asynchronous event handlers cannot be used when communicating with external databases.
- You cannot use the external database and tables for any event action where the workflow must be suspended. See the *Guide to the Application Event System*.
- User-defined fields cannot be used when communicating with external databases.
- All external tables referenced in an IDO must exist in the same database.
- RowPointer and RecordDate properties are used by Mongoose but do not exist in external 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.
- Due to driver limitations, these data-types are not supported for linked tables in a Progress database: LVARBINARY, BLOB, LVARCHAR, CLOB, BINARY, VARBINARY, RECID, TINYINT, BIGINT.

- For Postgres databases, the BIT data-type is not supported because IDO Properties have no equivalent data-type.
- Due to security requirements, you can perform read and write operations on a linked table in a DB2, Oracle, Postgres, or Progress database only through a configuration, where the **IDO Extension Classes Run In Partial Trust** flag is cleared.
- Due to driver limitations, you can perform write operations on a linked table in a DB2 or Progress database only through a configuration, where the **Use Distributed Transactions (DTC)** flag is cleared.

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 on page 21.

```
     public int IdoLinkOtherDbColPopulateSp( string linkedDatabase, string
 tableName, string optimisticColumnName, byte databaseType, string profile
Name, 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.CreateOutriggerAp
plicationDB( 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_PRE
CISION 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**.

Note: You may check in multiple IDOs on the **Mass Check In** form.

Checking in Multiple Objects

Note: Click **Refresh** on any tab to ensure that you are viewing the latest records.

To check in multiple objects at once:

1 Open the **Mass Check In** form.
2 On each tab, specify the files to check in by selecting the **Inc** option for each row, or by clicking **Select All**.
3 Optionally, to attach and check-in assembly images or symbols with an IDO extension class assembly:
   a On the **IDO Assemblies** tab, select the row of the IDO extension class assembly.
   b Click **Assembly Image** or **Symbols**.
   c Specify the image or symbol file to attach.
   d Click **OK**.

   Note: After importing an assembly image, or symbol, on the **Mass Check In** form, the collection is saved automatically.
4 Click **Check In**.
5 If your source control application requires it, specify any additional required information, such as comments or tracking numbers.

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 Click **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).

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,specify this information:
   - The IDO project with which this IDO is associated
   - The name of the primary base table
   - An alias by which the primary base table is to be known
   - The name of the IDO
3. Click **Next**.

   Note: If you specify a value for the Extends attribute, page 2 of the wizard is not used and the **Next** button is disabled. In this case, click **Finish**.
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**.

The new IDO is displayed, automatically checked out to you.

Setting Advanced IDO Attributes

The advanced attributes of an IDO include its primary key properties, and other options.

Primary Keys reflect the primary key properties of the IDO's base table in the application or external database. The system automatically detects primary key properties except for IDOs based on an SQL view. The order of the IDO's primary key properties should match the order used in the application underlying 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.
2. Click **Check Out**.
3. Click **Advanced Attributes**.

4 In the **Advanced IDO Attributes** form, the **Primary Keys** list indicates the primary keys detected by the system and their sort order. The **IDO Properties** list shows the properties of the IDO. Use these steps to work with primary keys:

- To add a property as a primary key, select the property 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.

Other advanced attributes control the behavior of IDO operation.

Attribute	Description or valid values
Quote Table Aliases	This flag controls whether appropriate quoting is applied to table aliases during SQL generation for the tables in this IDO. Set this flag to protect any table alias that represents an SQL keyword or reserved word in the query language of the server hosting the table’s database. If you select all table aliases to avoid duplicating SQL key-words, this flag can be cleared.

Editing IDOs

After you make changes in the application database, for example, modification of 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 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 might 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.
4. Select **Actions > Delete**.
5. Save your changes.

IDO Deletion Rules

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

- 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 these 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 this information:

- 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 IDOs

To export an IDO for use in a different Objects database:

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.

   Note: In a partial trust situation or in the web client, there is no **Browse** button. In these cases, you have a couple options to specify the path and filename:

   - You can enter a full path and filename to a directory on your local computer. For example: **`C:\Temp\MyIDO.xml`**
   - You can specify just a filename, in which case the system either prompts the user for a save location, or saves the file to the user's default download location.
2. In the **IDOs to Export** area, select the desired IDO project from the **Project Name** drop-down list. Then select one of these options:
   - To export all IDOs in the current project, select **All IDOs in selected project**.
   - To export only some IDOs, select **Selected** and then select the IDOs you want from the list.
3. Optionally, you can select property classes and IDO extension class assemblies to export.
4. When you have selected all the objects to export, click **OK**.

Exporting IDO Extension Class Assemblies

To export an IDO extension class assembly for use with a different Objects database:

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.

   Note: In a partial trust situation or in the web client, there is no **Browse** button. In these cases, you must enter the full path and filename manually. The path must be to a directory on your local computer. For example: **`C:\Temp\MyIDOExtensionClassAssembly.xml`**
2. In the **IDOs to Export** section, select the IDO associated with the assembly you want to export.
3. In the **IDO Assemblies to Export** areas, select one of these options:
   - To export all IDO extension class assemblies in the current Objects, select **All**.
   - To export only IDO extension class assemblies used by the IDOs selected in the **IDOs to Export** section, select **Referenced by Selected IDOs**.
   - To export only some IDO extension class assemblies, select **Selected**. Then select the assemblies you want from the list.
4. When you have selected all the objects to export, click **OK**.

Exporting Property Classes

To export a property class for use with a different Objects database:

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.

   Note: In a partial trust situation or in the web client, there is no **Browse** button. In these cases, In these cases, you have a couple options to specify the path and filename:

   - You can enter a full path and filename to a directory on your local computer. For example: `C:\Temp\MyIDO.xml`
   - You can specify just a filename, in which case the system either prompts the user for a save location, or saves the file to the user's default download location.
2. In the **IDOs to Export** section, select the desired IDO.
3. In the **Property Classes to Export** section, select one of the these options:
   - To export all property classes in the current Objects database, select **All**.
   - To export only property classes used by the IDOs selected in the **IDOs to Export** section, select **Referenced by selected IDOs**.
   - To export only some property classes, select **Selected**. Then select the property classes you want from the list.
4. When you have selected all the objects to export, click **OK**.

Importing IDOs

To import an IDO that was exported from a different Objects database:

1. On the **IDO Import Wizard** form, click **Browse** and specify a directory path and a filename for the file to be imported.

   The IDOs contained in the file you identified are displayed.

   The file must have been created with the IDO Export Wizard, and the file type must be XML.

   Note: In a partial trust situation or in the web client, there is no **Import File Name** field or **Browse** button. Instead, you must click **Import XML** and use the **Import Binary Data From File** dialog box to locate and upload the XML file you want from your local computer.
2. In the IDOs **IDOs To Import** section, select one of these options:
   - To import all IDOs contained in the XML file, select **All**.
   - To import only some IDOs, select **Selected** and then select the IDOs you want.
3. When you have selected all the objects to import, click **OK**.

Importing IDO Extension Class Assemblies

To import an IDO extension class assembly that has been exported from another Objects database:

1. On the **IDO Import Wizard** form, click **Browse** and specify a directory path and a filename for the file to be imported.

The IDO extension class assemblies contained in the file you identified are displayed.

The file must have been created with the **IDO Export Wizard**, and the file type must be XML.

Note: In a partial trust situation or in the web client, there is no **Import File Name** field or **Browse** button. Instead, you must click **Import XML** and use the **Import Binary Data From File** dialog box to locate and upload the XML file you want from your local computer.

2 In the IDOs To Import section, select one of these options:
- To not specify any particular IDO for the import action, select **None**.
- To specify all IDOs within the XML file, select **All**.
- To select only certain IDOs to include in the import action, select **Selected** and then select the IDOs you want from the list that displays.

3 In the **IDO Assemblies To Import** area, select one of these options:
- To import all IDO extension class assemblies contained in the XML file, select **All**.
- To import only some IDO extension class assemblies, select **Selected** and then select the assemblies you want.

4 When you have selected all the objects to import, click **OK**.

Importing Property Classes

To import a property class from a different Objects database:

1 On the **IDO Import Wizard** form, click **Browse** and specify a directory path and a filename for the file to be imported.

The property classes contained in the file you identified are displayed.

The file must have been created with the **IDO Export Wizard**, and the file type must be XML.

Note: In a partial trust situation or in the web client, there is no **Import File Name** field or **Browse** button. Instead, you must click **Import XML** and use the **Import Binary Data From File** dialog box to locate and upload the XML file you want from your local computer.

2 In the **Property Classes To Import** section, select one of these options:
- To import all property classes contained in the XML file, select **All**.
- To import only some property classes, select **Selected** and then select the property classes you want.

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, specify this information:
   - Table Name
   - Table Alias
   - Table Type
   - Join Type

5 If you are adding a secondary table, specify the join conditions.
6 Click **OK**.

The table and join conditions are added to the IDO.

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 **Tables** to display the **IDO Tables** 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 tool bar 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 specify 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 Table** 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 Table** or **New Table** form is displayed.
2 Specify your join conditions as a SQL SELECT statement in the large edit box. If necessary, you can use the three **Join Conditions** fields 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 might 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.

Non-primary base tables function as extensions to the primary base table. They must have the same key columns as the primary base table. The non-primary base table may store either of these types of data:

- Optional data, where a corresponding record might or might not exist for each record in the primary base table.
- Required data, where a corresponding record must exist for each record in the primary base table.

If a non-primary 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 .NET Framework class libraries that are created using 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.

1. Open the **IDOs** form.
2. Select an IDO and click **Check Out**.
3. Click **New Method**.
4. In the **Method Type** field 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 Select the **Stored Procedure** from the list of stored procedures in the application database.
6 In the **Method Name** field, either select the method want from the drop-down list, or enter the name of the new method.

   The method name must be unique to this IDO.
7 If the method should be run as a transaction, select **Transactional**.
8 Click **OK**.

After you refresh, the new method displays 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 Specify 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 or property from an IDO:

1 In the **IDOs** form, select the IDO you want.
2 Click **Check Out**.
3 Depending on whether you want to delete a method or a property, click either **Methods** or **Properties**
4 Select the method or property to delete.
5 From the **Actions** menu, select **Delete**.
6 Save your changes.

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

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

Adding or Editing Validators for a Property or Class

You can add, edit, or change the order of validators for a property or property class.

1. To access the **Edit Validators** form, click the ellipses (...) button to the right of the **Validators** field in either the **IDO Properties** form or the **Property Classes** form.

   Note: Any validators you define here are executed by the IDO Runtime service during a collection save operation. These validators are also inherited by any component on a form that is bound to this property or to a property derived from that class.
2. On the main page of the form, click **Add** to add a validator.
3. On the **Select Validator Type** page, use the option buttons to select the type of validator to add:
   - **`In Inline List`**
   - **`In IDO Collection`**
   - **`IDO Method`**
4. Click**Next**. The page you go to is determined by your choice on this page.

5 The **Set Properties From Inline List** page displays in these cases:

- You select **In Inline List** on the **Select Validator Type** page and then click **Next**.
- You select an **In Inline List** validator on the main page and then click **Edit**.

Use this page to select additional properties to be set from the inline list on a successful validation. On this page, use the grid to specify:

- The property to be updated if validation is successful (**Property Name**)
- The column from the inline list from which the value is to be taken (**Inline List Column**)

You can also optionally specify an error message to display if the validation fails.

6 The **In Collection Validator** page displays in these cases:

- You select **In IDO Collection** on the **Select Validator Type** page and then click **Next**.
- You select an **In Collection** validator on the main page and then click **Edit**.

Use this page to select the IDO and IDO property that are to be set on a successful validation. On this page, use the grid to specify:

- The target property; that is, the property for which the new value is to be assigned if the validation is successful
- The source property; that is, the property to be evaluated during the validation activity

You can also optionally set a filter and any additional properties to be set if the validation is successful.

Finally, you can also specify an error message to display if the validation fails.

7 The **IDO Method Validator** page displays in these cases:

- You select **IDO Method** on the **Select Validator Type** page and then click **Next**.
- You select an **IDO Method** validator on the main page and then click **Edit**.

Use this page to specify the IDO and method that are to be used to validate the property value. Specify any parameters that the method requires in the grid. These parameters can be:

- Input only, both input and output, or message values (**Type**)
- Either property values or literal values (**Source**)

The validator then evaluates the **Value** at run time. If the **Source** specifies a property value, then the **Value** is treated as a property value. if the **Source** specifies a literal value, then the **Value** is treated as a literal value.

Finally, you can also specify an error message to display if the validation fails.

8 On the main page, to edit a selected validator, click **Edit**.

When there are multiple validators, you can rearrange the order in which they validate using the **Move Up** and **Move Down** buttons.

9 To close this form and return any validator definitions to the parent form, 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.

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.

For example, suppose you want to create an inline list with this information:

- 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, this metadata is created 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.

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**.

The page you proceed to depends on the type you selected.

5 If you selected `In Inline List`, follow these steps:
   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 `In IDO Collection`, follow these steps:
   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`, follow these steps:
   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**.

Chapter 3: 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 objects 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 any operation is made on objects, Form Control checks the master and run-time databases for changes, and flushes the cache before proceeding with any operation.

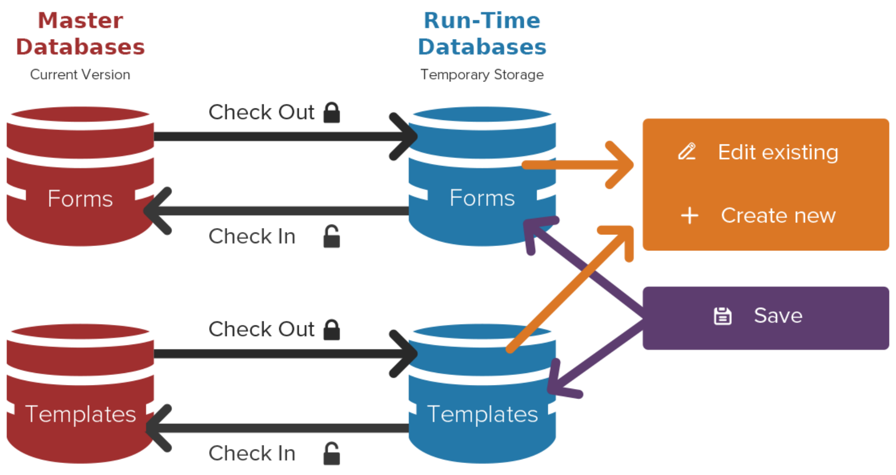
Before You Begin

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 this information in mind:

- 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, even if the object is locked to a developer. 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 work on certain forms or global objects in a development environment. Form Control is limited to these basic tasks:

- Checking objects out on page 45
- Checking objects in on page 47
- Getting objects on page 47
- Unlocking objects on page 48
- Archiving objects on page 48
- Restoring objects on page 49
- Displaying status of objects

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 these objects:

- Forms, with or without their associated templates and/or global objects
- Global objects, regardless of whether they are associated with forms

During a checkout procedure, Form Control performs these tasks:

- 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.

Displaying and Maintaining Checked-Out Objects

You can display a list of everything you have checked out, and you can unlock or check in items from the list. This display can include only certain types of objects, such as forms, or it can display all objects. This list can also include objects that you have created but not yet checked in.

Once the list is displayed, you can select which objects you want to take further actions with. Those further actions can include checking in selected (or all) objects or unlocking them without checking them in.

Displaying Your Checked-out Objects

To display your checked-out objects:

1. In the **Form Control** form, select the **My Checked Out Objects** tab.
2. Click **Display List**.

   In the **Checked Out Objects** pane, all the objects you have checked out are listed. The values, if any, for each object, both in the master and run-time databases, are displayed along with the name and type of each object.

   Note: This display does not include any template objects you might have checked out. Forms and global objects that you have created are displayed in this list only if you selected **Enforce strict locking of forms and objects** in the user settings, and you select **Include Newly Created** on this tab.

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 the object checked out.
- The object is new and does not exist in the master database.

You can check in these objects:

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

During a check-in operation, Form Control performs these tasks:

- 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.

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 these objects:

- Forms, with or without their associated templates and/or global objects
- Global objects, regardless of whether or not they are associated with forms

During a get operation, Form Control performs these tasks:

- 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.

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.

Note: You can unlock objects that you have previously checked out and objects that others have checked out. Other developers can also unlock objects that you have checked out.

You can unlock these objects:

- Forms, with or without their associated templates and/or global objects.
- Global objects, regardless of whether they are associated with forms.

During an unlocking operation, Form Control unlocks the objects in both the master and run-time environments. 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.

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 these objects:

- Forms, with or without their associated templates and/or global objects
- Global objects, regardless of whether or not they are associated with forms

During an archive operation, Form Control performs these tasks:

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

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 these objects:

- Forms, with or without their associated templates and/or global objects
- Global objects, regardless of whether they are associated with forms

During a restore operation, Form Control performs these tasks:

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

Chapter 4: Application Events

About the Application Event System

The application event system is a powerful and flexible set of tools that is designed to help extend and customize how the system works. You can use these tools to define and monitor application events across the system, rather than being limited to a single form.

The application event system is extremely powerful and flexible. However, for some non-developer users, the application event system can be overwhelming at first. To help you with the learning curve, Infor provides a **New Workflow Wizard**, which is designed to step you through the process of creating simple workflows using the application event system.

Some Mongoose-based applications also include a **Workflow Event Handler Activation** form to help non-developers create simple application event system workflows. This form provides access to a set of predefined workflow event handlers. These handlers are ready-to-use. In most cases, you must define the user names or email addresses of the people who must be notified when a certain event happens, and activate the handler.

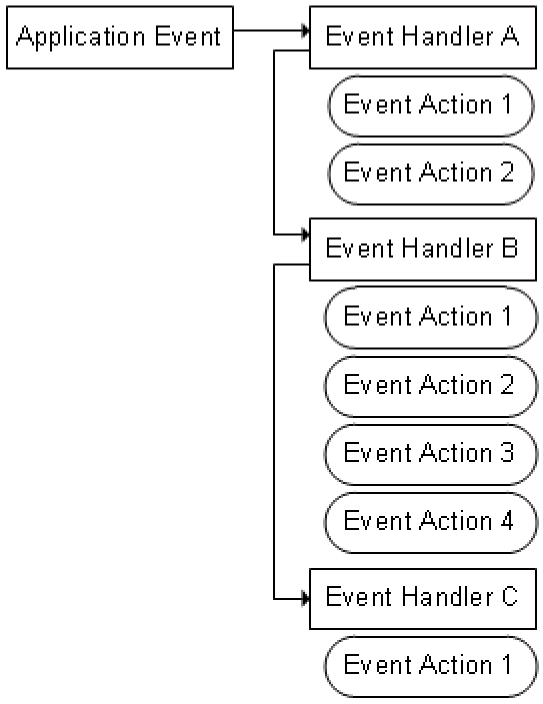
Basic Components of the Application Event System

The application event system consists of interrelated components:

- Application events are uniquely named incidents that can occur during the use of an application. These events can have multiple triggers and can be generated by user actions, conditions in a database, other events, or other situations.
- Event handlers consist of data that specifies:
  - The application events to which they are to respond
  - Any conditions, situations, or attributes that determine when and why each handler executes
  - One or more event actions to take place during the handler’s execution

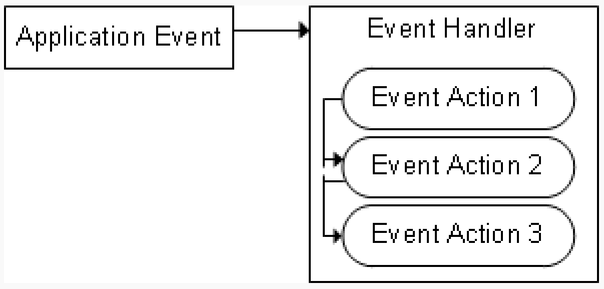
  Each application event can have multiple handlers, but each handler can be associated with only one application event.
- Event actions consist of instructions that specify the individual tasks or bits of work that are performed by the event handler. Each event handler must have at least one action and can have multiple actions.

This diagram shows a possible set of application events, event handlers, and event actions:



You want the system to automatically notify you whenever someone adds a customer order to the system and to request your approval if the order is $1000 or more.

This diagram shows the application event and associated handlers for this scenario:



The application event gets triggered whenever someone creates a new customer order. The application event that tells the event handler to run are set up as part of the event handler definition. The event action is a notification that an order has been placed and an approval request for orders of $1000 or more.

A single event handler is required, but that event handler requires these event actions to complete all the requirements:

- Event Action 1: The system checks the amount of the order and determine what additional actions are required based on the amount.
  - If the amount is $1000 or more, go to Event Action 2.
  - If the order is less than $1000, go to Event Action 3.
- Event Action 2: The system generates a prompt message to the manager that requests for an approval for the order. The system then waits until the manager responds.
  - If the manager approves the order, the system then proceeds to the next action.

  - If the manager disapproves the order, the event handler fails and no further action is taken.
- Event Action 3: The system sends a message to the inbox with a notification to let the manager know that the order has been completed.

Advantages of the Application Event System

Application events and event handlers are tagged at the time of their creation with a special identifier (Access As) that prevents them from being modified or deleted by other development organizations. This identifier also prevents your custom events and handlers from being overwritten when Infor or another developer issues updates to their events and handlers.

The application event system provides an infrastructure that allows both the framework and application code to generate application events, and for developers and system administrators to create handlers for those application events. Event handlers can augment or replace the default framework or application behavior associated with the application event. You can design event handlers so that maximal work can be performed without requiring new procedural code.

Event handlers are defined using metadata. Metadata refers to the practice of using uncompiled code and information about data formats that are interpreted during run time, rather than compiled code or procedural code.

The application event system offers several significant advantages over hard-coded systems:

- The application event system provides the power and flexibility to customize the way the system performs without modifying the basic code. Therefore, business processes are easier to modify. You can modify application processes and policies without directly modifying the application code and without writing any procedural code. This means that the amount of procedural code that is required to implement functionality can be greatly reduced or even eliminated altogether.
- You can use your event handlers with events created by others to gain control of the application flow and take appropriate action, rather than being forced to call into the application using APIs.
- Because event handlers are defined using metadata, there is no upgrade problem, and no collision problem if other developers also have event handlers for the same event.

Example: Using the Application Event System

These examples offer illustrations of how to use the application event system, including explanations of how this can benefit you.

Example 1: Sending a notification when a record is added

Scenario: You have a sales manager who wants to be notified whenever someone adds a new customer to the system.

Solution: You can use the application event system to automate compliance with this request. You can set up an application event that is generated whenever anyone adds a new customer into the system, and use event handlers and actions to automatically generate a notification that is sent to the sales manager.

Example 2: Getting approval for a credit limit change

Scenario: Your company requires that any change to a customer's credit limit be approved by a designated credit manager.

Solution: You can use the application event system to automatically send the credit manager a message requesting approval of the change. To speed up the process, you can also send an email to the credit manager that contains links that the manager can click to approve or deny the request. When the credit manager responds to the message and approves the request, the system can then automatically change the credit limit amount.

Example 3: Complex approval of a purchase order status change

Scenario: Your company has a business process in place that:

- Requests approval from a purchasing manager when the status of any purchase order (PO) is changed to **`Ordered`**
- Requests majority approval from a group of higher level managers when the total cost of the PO exceeds a certain amount and whether the order should be sent to the vendor
- Requests unanimous approval from a group of top-level directors when the total cost of the PO exceeds a higher amount
- Approves and Completes the transaction when the total cost of the PO is approved at all the required levels

Solution: You can use application events and handlers to automate this entire process. You can evaluate the PO at each step, request only the necessary approvals, and complete the transaction upon approval.

Example 4: Automatically shipping a customer order

Scenario: You have a system that, by default, requires someone to manually set the system to ship a customer order line when the status of the line is set to **`Filled`**.

Solution: You can use the application event system to automatically set the order to ship as soon as the status is set to **`Filled`**.

Using the Application Event System for Document (File) Attachments

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

- Include documents attached to records when initiating those records for processing
- Attach, link, or detach documents to records during the course of processing

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 Email 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.

With the application event system, you can update or detach documents that are already attached to records and included in such actions. For more information about how to use the fields, see the help on the Event Action Attach form.

Processing 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.

Working with Application Event System

Application events are generated or fired in response to an action or condition that occurs in the system. When the application event is generated, it can execute one or more event handlers with the event actions associated with each event handler.

Handling Application Events

An application event requires an event handler to do an action in response. Otherwise, the application event serves no practical purpose. If there is no handler for an application event, the event does nothing when generated.

Note: Some events can exist without an event handler. Infor includes framework events as part of the system. Infor does not include handlers for these events. These events were created to provide events that other developers can use with their own custom event handlers.

When application events can be generated

An application event can be generated when:

- A system user performs an action on a given form or when a business process is involved
- A database calculation is performed that results a certain value
- Another event results in generating this application event
- An amount of time has passed

These situations and conditions can fire application events:

- A sales representative saves a record in the **Customers** form
- A manager changes the credit hold status of a customer
- A factory manager adds a new item to the list of manufactured items in that facility
- The first day of each month arrives, for example, an application event can be used to generate a monthly report
- The quantity-on-hand of an item is less than zero

Where application events can be generated

With the application event system, application events can be generated from any tier.

- In the client tier, the application event can be generated by using a form that has a form event handler with a **`Generate Application Event`** response type.
- In the middle tier, the application event can be generated by invoking an appropriate .NET method.
- In the database tier, the application event can be generated by using an appropriate stored procedure.
- In any tier, an application event can generate another application event by using the **`Generate Event`** action type.

Controlling the Sequence of Event Handlers and Actions

These settings control the order in which event handlers and their actions execute.

Handler sequence number

Each event handler has a handler sequence number. Handlers execute in the order of their sequence numbers. To modify the flow and change handler sequence order, use the **Keep With** and **Chronology** options on the **Event Handlers** form or the **Event Handler Sequence** form.

Action sequence number

The event actions associated with each event handler have their own action sequence numbers. These numbers execute in numeric order, unless:

- You use the **Initial Action** option on the **Event Handlers** form to designate that an action should execute first
- You use event action types to modify the flow

Restricting Which Handlers Run

There may be times when you want or need to disable the event handlers created by one or more development organizations, including yours, at least temporarily. This occurs typically when you are troubleshooting problems with the application event system.

You can disable event handlers by using the **Event Handlers** form or the **Session Access As** form.

- To disable event handlers temporarily, on **Event Handlers** form, clear the **Active** check box.

  Note: You cannot use this technique to disable Core event handlers.

- To disable (or enable) all the event handlers that are created by a developing organization all at once, use the **Session Access As** form.

  Note: To accomplish the same thing for individual handlers with your **`Access As`** value, open the **Event Handlers** form. For each event handler that you do not want to include in testing or debugging, clear the **Active** check box.

For example, a customer has a problem that he suspects is caused by something that he did in the event system, but he is not sure what. He places a call to Infor Technical Support.

The technical support representative verifies that the customer's custom event handlers are not causing the problem. The technical support representative asks the customer to temporarily disable all custom event handlers so that the operation can be tested with only standard functionality in place. The technical support representative instructs the customer to use the **Session Access As** form to perform either of these actions:

- In the Include **Access As** field, specify **`Core,BaseAppAccessAs`**, where *BaseAppAccessAs* is the Access As identifier associated with the base application installed on the system.
  Note: This option allows only the Infor framework and base application event handlers to execute. Custom event handlers that are created by the end-customer do not execute.
- Leave the **Exclude Access As** field blank, and select the **Exclude Blank Access As** check box.
  Note: This option allows all Infor and business partners event handlers to operate. Only the customer's event handlers are ignored.

To disable or enable event handlers using the **Session Access As** form, use any of these options:

- In the **Include Access As** field, specify the Access As identifiers for event handlers that are recognized during this session.
  Note: To include multiple Access As identifiers, specify them and separate only by commas (no spaces).

  If this field is left blank, the system can recognize and execute all event handlers.

- In the **Exclude Access As** field, specify the Access As identifiers for event handlers that are ignored during this session.
  Note: To exclude multiple Access As identifiers, specify them and separate only by commas (no spaces).

  If this field is left blank, the system can recognize and execute all event handlers. The exception occurs only if the **Exclude Blank Access As** check box is selected. In this case, all event handlers are recognized except event handlers for which the Access As identifier is null (blank).

- Select the **Exclude Blank Access As** check box to exclude the event handlers that have a blank Access As identifier.

About Synchronicity

With respect to the application event system, synchronicity refers to the ability or inability of event handlers to run independently from other event handlers. Application events are synchronous when they must execute their handlers in a specific order. Application events are asynchronous when they can execute their handlers independently of other handlers.

A synchronous event handler is one that must be complete before the system continues to the next event handler in a sequence. For the entire application event to be handled successfully, all synchronous handlers in the sequence must complete successfully. The exception to this rule is that, if one event handler fails, other than for an illegal operation, and the system is set to ignore failures for that handler, the system can continue to the next synchronous event handler. Otherwise, the system returns a failure error, and no more event handlers in the sequence execute.

By contrast, an asynchronous event handler runs independently of other event handlers.

Synchronous events

Unless the application event is designed to suspend, the expectation is that the application event is completed synchronously. Therefore, synchronous event handlers execute in sequence in the same thread that generated it and block that thread until they are all executed or until a handler exits with a failure status.

Application events are synchronous when they are generated by:

- Framework events
- Calling the FireApplicationEvent() .NET method with the **Synchronous** parameter set to True
- Using a form event handler with a **`Generate Application Event`** response type and the **Synchronous** option selected
- Using a **`Generate Event`** event action type with the **Synchronous** parameter set to True

Asynchronous events

This application event runs in a different thread than the one that posted it, usually on an application (utility) server. In this case, the application event does not block the generating thread, running independently of it. As soon as an acknowledgment is received that the event was successfully queued, the thread that generated the application event continues.

Application events are asynchronous when they are generated by:

- Calling the FireApplicationEvent() .NET method with the **Synchronous** parameter set to False
- Calling the dbo.FireEventSp or dbo.PostEventSp stored procedure
- Using a form event handler with a **`Generate Application Event`** response type and the **Synchronous** option cleared
- Using a **`Generate Event`** event action type with the **Synchronous** parameter set to False

Event handlers

Event handlers can be designated as synchronous or asynchronous at the time they are created, using the **Synchronous** check box on the **Event Handlers** form.

Any event, either synchronous or asynchronous, can execute an asynchronous event handler when execution reaches an event handler designated as an asynchronous event handler; that is, for which the **Synchronous** check box is cleared. At this point:

- The system sends the asynchronous event handler to the event handler queue.

Note: The system performs queueing by means of the PostEventHandlerSp stored procedure, to which it passes the configuration name from the event state.

- If queueing was successful, the application event's thread continues to the next event handler or, if no subsequent handlers are defined, completes the event.
- If queueing was unsuccessful, the application event stops with a failure condition.

About Suspension

Suspension occurs when a requested operation is sent to the application event system for completion at a later time. This occurs when the event handler has the **Suspend** option selected and contains an adjourning action.

Suspension is possible only with certain framework events. You cannot create custom events that can be suspended. Currently, these application events can be suspended:

- IdoOnItemInsert
- IdoOnItemUpdate
- IdoOnItemDelete

Suspension occurs when the system generates an application event that:

- Is one of these framework events that can be suspended
- Has the **Suspend** check box selected (on the form) for at least one handler that applies to the generated event’s object and initiator

When an application event is suspended

When both conditions are met, WinStudio passes control of the requested update (insertion, update, or deletion) to the application event system. The application event system then ensures that the event handlers can all execute successfully before actually committing the system and the data to execute the event actions.

There are two stages: suspend-validating stage and suspend-committing stage.

When an application event is not suspended

When an application event that can be suspended is not suspended (that is, no event handler for that application event that applies to that event’s object and initiator has the **Suspend** check box selected), the system executes the application event using this process:

1. The system performs a database update.
2. The application event system executes all effective event handlers, returning either success or failure.
3. The system handles failures as errors and exits the application event.

Note: Each insertion/update/deletion request in the process, which could be from an external XML request or from a Save action in WinStudio, participates in the overall transaction. In this case, all complete or nothing completes. So, each handler chain (the same chain executes for each insertion, update, or deletion in the group) also participates in that overall transaction. A failure of any of those handler chains results in the entire transaction being rolled back.

About Suspension Stages

Suspension of application events occurs in these stages:

Suspend-validating stage

When an application event is suspended, in what is known as suspend-validating stage, the system:

1. Begins a database transaction.
2. Performs an update to the database to validate pertinent data against any SQL constraints or triggers.
3. Validates the data by running all effective event handlers that are synchronous and not suspended in the current transaction.

   This process involves both application and event system changes:

   - At the beginning of this process, in a separate event transaction, the system copies the event handler to a new revision, if necessary.

     The reason for this is that both the suspend-validating mode and the suspend-committing mode must use the same metadata. However, the validating transaction will be rolled back, and the master transaction may be rolled back. So, the current event handler revision must be available, in case someone edits the handlers between execution of the two modes.
   - During this process, a non-output event parameter named **Suspend_ValidatingMode** with a value of 1 is made available to handlers.

     You can test this on any synchronous, non-suspended handlers that you do not want to execute at this time (perhaps because its actions would irreversibly affect something not controlled by our database transaction), using the expression **`CONDITION( E(Suspend_ValidatingMode)<>1`**).

     Note: At this point, the application event system treats any failure as an error and exits the execution of the application event.
4. Rolls back the transaction
5. Places a task on the event queue to run all involved handlers in suspend-committing mode.

   If the collection update is rolled back due to a later error, this suspension is also rolled back with it and results in a state identical to one in which this update was never requested.

   When the application event service picks up the queued event, it knows that a suspension is in effect by checking the **EventQueue.Suspend** attribute.
6. The system marks the record involved in the update/deletion request to have the InWorkflow property set to 1. This prevents any user from attempting another change to the suspended record. Note that suspended insertions do not reach the database at all unless and until the suspended event finishes successfully.
   Note: Suspended records in a workflow are indicated as such by a purple icon in the status indicator column of a grid.
7. The thread in which the event was generated continues as if the change request succeeded. This occurs even if the thread originated from an event handler executing an **UpdateCollection** or **ExecuteIDORequest** type event action. In other words, suspension affects only one level of event execution.

Suspend-committing stage

Once the suspended event completes the suspend-validating mode successfully and is placed on the event queue, the system processes the event using these rules. This is known as the suspend-committing stage.

- Any failure or error condition prevents further event handlers to execute and prevents the standard processing (requesting for insertion, update, or deletion) to occur. However, the **InWorkflow** attribute is cleared.
- After each event handler finishes successfully, execution proceeds to the next event handler.
- The system queues any asynchronous event handler and execution proceeds to the next event handler.
- When the last event handler is finished, or is queued successfully, the standard processing is committed and the application event is finished.
  Note: For an update request, the **InWorkflow** attribute is cleared.

About payload

An application event that can be suspended carries a payload that represents and allows handlers to access and modify the requested change (insertion, update, or deletion).

Each named property value that forms part of the change is passed into and out of the application event system as a parameter with a name of the **`Row.Property`** form, where *Property* is the name of the IDO property. Modified property values are accompanied by another parameter with a name of the **`Row.Property.Modified`** form and a value of 1.

During notify and prompt actions, these parameters are:

- Temporarily converted to event variables, which overrides any value that may have been set in preceding actions or specified on the **Event Variable Groups** form
- Available for display (and update, for prompts) on the **Variables** tab of the **Inbox** form (and for display at the end of an external email prompt)
- Then converted back to parameters for subsequent actions

During other actions, such as Set Values, these parameters can be modified to affect the result of the change when it is applied to the database. This can be done by using this syntax:

- **`SETPROPVALUES("Property"=expression)`**: If the new value is different than the original framework event parameter received by the event, the PROPERTYMODIFIED(PropertyName) framework event parameter is also set to True automatically.
- **`SETPARMVALUES(Row.Property=expression, Row.Property.Modified="1")`**: Note that you should also indicate to the framework that the property is now modified, as shown.

About Adjournment and Resumption

Some chains of event actions must be temporarily stopped while waiting for some condition to be met. For example, a customer order needs a manager’s approval before the order can be entered into the system. In this case, the system temporarily halts the execution of the event handler chain until the manager responds with an approval. This event operation is known as adjournment.

Adjourning actions must wait for an external stimulus before the event handler can proceed with the next action. An event handler that contains such an action must be either asynchronous or part of a framework event that can be suspended and is marked to suspend.

When execution reaches an adjourning action, the event handler state is set to retest or time out at a specified time. At this point, the application event becomes an adjourned event.

The event service then processes the application event at the next opportunity after the Time Out or Retest time, or both. This event operation is called resumption.

About Event Handler Status

Each event handler can exit with a Success or Failure status. When an event handler fails, if it is not set to ignore failures, and if it has not attempted an illegal operation, the system can retry the event handler's actions.

Success

An event handler exits with a Success status only when it reaches a **`Finish`** event action or completes all event actions successfully without reaching a **`Fail`** event action.

Failure

An event handler exits with a Failure status when it does any of these:

- Reaches a **`Fail`** event action
- Attempts an illegal operation
- Experiences an unexpected failure of an event action or something launched by an event action

When a failure occurs:

- In the **Result** field of the **Event Handler Status** form, an appropriate error message is displayed, which includes an error message passed from a failing stored procedure in @Infobar message type parameter, where applicable.
- On the **Actions** tab of the **Event Handler Status** form, the **Times Failed** counter for the current event action state is incremented.
- On the **Handlers** tab of the **Event Handler Status** form, the event handler state's **Last Activity Date** is set to the date and time of the failure.
- On the **Variables** tab of the **Event Handler Status** form, any event variables are maintained at their current values for inspection.

Ignoring Failures

By default, if any synchronous event handler reports a failure, the system skips any remaining event handlers for that application event and exits with a Failure status.

To override this default behavior, on the **Event Handlers** form, select the **Ignore Failure** check box.

If you select this option, the event handler is always treated as successful unless an illegal operation is deliberately attempted. However, in the **Status** field of the **Event Handler Status** form, the status remains Failure, even though the **Any Handlers Failed** check box is cleared on the **Event Status** form.

This table shows what conditions can result in failures for various action types and whether the failure can be ignored.

Action type for the current event action	Condition	Ignore Failure option selected?	Result (Event Status)
Set Values	Illegal operation: Any attempt to set a non-output event parameter	N/A	Failure
Some	Illegal operation: Any attempt to use an action type that is not supported on the current tier	N/A	Failure
Any	Illegal operation: Syntax error in parameter	N/A	Failure
Generate Event	Illegal operation: Any attempt to generate a framework event	N/A	Failure
Any	Unexpected error	No	Failure
Any	Unexpected error	Yes	Success

A Fail action whose condition evaluates to True on an event handler for which the **Ignore Failure** option is selected is equivalent to a Finish action.

An asynchronous event handler's failure does not affect any remaining event handlers for that event. However, setting the **Ignore Failure** option in this case avoids affecting the AnyHandlersFailed() attribute of that event, unless the failure was caused by deliberately attempting an illegal operation, as shown in the table.

Retrying Event Handlers

On the **Event Status** and **Event Handler Status** forms, you can retry a failed event handler, if it is retryable. Whether a handler can be retried is determined by the cause of the failure and whether the event handler is marked as transactional, as shown here:

Cause of Failure	Event transactional	Handler transactional	Retryable	Retry point	Event Variables
Illegal operation	N/A	N/A	No	N/A	N/A
Any legal operation	Yes	N/A	No	N/A	N/A

Cause of Failure	Event transactional	Handler transactional	Retryable	Retry point	Event Variables
Any legal operation	No	Yes	Yes	InitialAction	Re-initialized
Unconditional Fail Event action	No	No	Yes	InitialAction	Re-initialized
Conditional Fail Event action	No	No	Yes	CurrentAction	Maintained
Unexpected error	No	No	Yes	CurrentAction	Maintained

When retrying a single handler, whether the handler runs synchronously or asynchronously depends on the parameter passed to the API, not the handler's definition. When retrying an event, the handlers are run as shown in this table:

	Failed Synch Handler	Failed Asynch Handler
Restarted synchronously	Synchronous	Asynchronous
Restarted asynchronously	Asynchronous	Asynchronous

An event can have a synchronous handler that is not retryable, but it still retries any asynchronous handlers that can be retried. In this case, any synchronous handlers that will come after the failed handler will not be run.

The Event or EventHandler Result must be cleared when retrying the handler. However, the previous handler result is saved in a handler variable called PreviousResult, and the event results are saved in an event parameter also called PreviousResult.

About Transactions

A transactional event is an application event or set of event handlers in which all must complete successfully before any data is committed in the system. If any handler fails, then the system rolls back or reverts all data to its initial state.

If the transactional event contains any adjourning event actions, the system modifies this behavior somewhat: Any actions taken up to the first adjournment, between each resumption of an adjournment and the beginning of the next adjournment, and after the last resumption, are each committed separately.

The system treats any asynchronous event handlers as if they belong to an asynchronously generated non-transactional event. In other words, because they are outside the synchronous flow, these event handlers are not treated as part of the transaction.

Transactions with synchronous events

An event generated synchronously without the **Transactional** check box selected on the **Event Handlers** form either:

- Runs in the current transaction state of the firing thread, if no lower-level transactioning is specified
- Handles transactions at the event handler or event action level, as explained in these points:
  - In the middle tier, any execution methods for database-related event actions must enlist in an existing transaction or create a new one.

    To include the entire event in a single transaction, call the FireApplicationEvent method from a hand-coded IDO method that is marked **Transactional** in the IDO metadata. In this case, the execution methods for the database-related event action enlist in the transaction created by the IDO Runtime for that transactional IDO method.

    When the FireApplicationEvent method is called from a hand-coded IDO method that is not marked **Transactional** and whose caller is not in a transaction state, each event action execution method creates a new transaction, which is rolled back in case of failure, except when the containing event handler is marked **Transactional**. Changes to event system state data are performed in a separate connection (except in suspend-validating mode), so they survive any rollback of the encompassing or wrapping transactions.
  - The client is not permitted to control transactions that span form event handler calls. Client- tier event-firing requests are passed directly on to the MGCore.Events.FireEvent IDO method.

A failure of a synchronous event returns an error condition to the firing thread. For framework events, this causes the entire current transaction to be rolled back. For application events, whatever generated the event (.NET method or VB.NET script) is responsible for trapping the error condition, accomplishing a rollback of the current transaction, and throwing execution to an appropriate recovery point in or out of that code.

Transactional event handlers

An event handler marked **Transactional** is wrapped in a new transaction when executed from the middle tier, if no encompassing transaction is active at the trigger point of the outermost enclosing event.

This event handler cannot contain any adjourning event action types (for example, Prompt, Wait, or Sleep), but it can be marked **Asynchronous** itself. If this event handler ends in a failure, the wrapping transaction is rolled back (assuming it was created).

Non-transactional event handlers

An application event that is generated asynchronously without the **Transactional** setting does not run in a transaction state. Event handlers that are not marked **Transactional**, for a synchronous event generated from a middle-tier non-transactional custom IDO method, are handled this way:

- An application event can be generated asynchronously with a **Transactional** setting that indicates that a new transaction has begun and the entire database-related effects of its synchronous event handlers are committed if they all succeed or rolled back if any fail.
- Any asynchronous event handlers are treated as if they belong to an asynchronously generated non-transactional event.

Rolling Back Transactions

These conditions cause the system to set the **Transactional** setting on a new event state row:

- FireApplicationEvent() detects an enclosing transaction.
- FireEventSp, FireApplicationEvent(), PostEventSp, or PostEvent() receives a **Transactional** setting.

This event state setting signals the system administrator that when this event state is finished and not rolled back, everything happened that was expected to happen.

In a synchronous multi-event situation, to encode the proper information in the **Rolled Back** setting, an enclosing transaction that contains one or more FireApplicationEvent() calls must:

1 Once, before calling FireApplicationEvent(), call EventBeginTransactionSp or EventBeginTransaction().

This uses a separate transaction to clear a storage area to record the row pointers of the event state rows to be created and returns a transaction identifier to be passed to FireApplicationEvent().

2 Pass that transaction identifier to FireApplicationEvent().

In a separate transaction, this records the row pointer of the new event state row.

3 Upon rolling back or committing the transaction, call EventEndTransactionSp or EventEndTransaction() and pass the transaction identifier and whether a commit or rollback is performed.

This uses a separate transaction to set the **Rolled Back** setting on the stored event state rows (if rolling back), and clears the storage area.

If this is neglected, the system administrator cannot determine why database actions that appear to have finished successfully according to the status forms are not reflected in the database.

In asynchronous situations and when a failure is detected by the event system, the **Rolled Back** setting for the failing transactional event is set by the event system.

Passing Parameters to an Asynchronous Event

Sample code:

```
DECLARE @EventParmId AS UNIQUEIDENTIFIER = NEWID()
EXEC InsertEventInputParameterSp @MyEventParmId, 'Parameter1', @Variable1
EXEC InsertEventInputParameterSp @MyEventParmId, 'Parameter2', @Variable2
EXEC InsertEventInputParameterSp @MyEventParmId, 'OutParameter3', NULL,
1
DECLARE @SessionID AS UNIQUEIDENTIFIER = dbo.SessionIDSp()
DECLARE @Unused AS BIT = 0
DECLARE @Result EventResultType
DECLARE @Infobar InfobarType
EXEC dbo.FireEventSp
   @EventName = 'MyEvent',
   @ConfigName = 'MyConfiguration',
   @Initiator = 'MyInititator',
   @SessionID = @SessionID,
   @EventParmId = @MyEventParmId,
   @EventTrxId = NULL,
   @Transactional = 0,
   @GeneratingEventActionStateRowPointer = NULL,
   @AnyHandlersFailed = @Unused OUTPUT,
   @Result = @Result OUTPUT,
   @Infobar = @Infobar OUTPUT
```

Calling a Synchronous Event within a Transaction and Handling Failure

1 Determine the current site.

2 Name a configuration, by convention:

```
DECLARE @Site SiteType
SELECT @Site = site FROM parms
```

3 Determine the current SessionId:

```
DECLARE @SessionId RowPointerType
SET @SessionId = dbo.SessionIdSp()
```

4 Add the procedure code:

```
BEGIN TRANSACTION
UPDATE coitem
SET due_date = dbo.CalcDueDate(@Parm1, @Parm2)
WHERE coitem.co_num = @CoNum
AND coitem.co_line = @CoLine
AND coitem.co_release = @CoRelease

SET @MyEventParmId = NEWID()
EXEC InsertEventInputParameterSp @MyEventParmId, 'CoNum', @CoNum
EXEC InsertEventInputParameterSp @MyEventParmId, 'CoLine', @CoLine
EXEC InsertEventInputParameterSp @MyEventParmId, 'CoRelease', @CoRelease

DECLARE
@anyHandlersFailed [tinyint],
@result [nvarchar](4000),
@Infobar [nvarchar](4000)

EXEC @Severity = FireEventSp
@eventName = SetCoitemDueDate',
@configName = 'SyteLine',
@sessionID = @SessionID,
@eventTrxId = null,
@eventParmId = @MyEventParmID OUTPUT,
@transactional = 0,
@anyHandlersFailed = @anyHandlersFailed output,
@result = @result output,
@Infobar = @infobar output

IF @Severity > 0
```

```
BEGIN
EXEC RaiseError @Infobar, @Severity
ROLLBACK TRANSACTION END
...
COMMIT TRANSACTION
```

About the Framework Event Service

The Framework Event Service is an independent process that can run on any application (utility) server. Each instance of the Framework Event Service monitors the event queue for any number of configurations defined locally.

For load-balancing purposes, this service safely acquires work from the database queue of each configuration's application, so that multiple event services can run on one or multiple servers. The event service monitors the event queue for these events and handlers:

- New events that have been queued by means of asynchronous generation methods or suspending framework events, as found in the Queued Event table
- New event handlers that have been queued by means of the firing of asynchronous event handlers, as found in the Queued Event Handler table
- Resuming event handlers that had been adjourned, and have now completed a Prompt, Wait, or Sleep event action, as found in the Event Handler Status table having a value in **Retest At Date** that is older than the current date and time
- Resuming event handlers that had been adjourned, and have now completed a Discover File action, as found in the Event Handler Status table having a value in **Path To Watch** and no value in **Last Watched Date**
- Running event handlers whose current action has timed out, as found in the Event Handler Status table having a value in **Times Out At Date** that is older than the current date and time
- Active event triggers
- Continuing events that have successfully retried a previously failed handler, as found in the Event Status table having a value in **Continue After Handler Row Pointer**

Administrative details

To eliminate the possibility of deadlocks when manipulating event data, a second database connection is used for event data modifications, that is, separate from the one used by the event action execution methods. Also, any event data modifications that do not need to be available to other processes (any tier of any interactive user session or other event service instances) are made in memory and written only when required.

To avoid overtaxing the application server, the Framework Event Service administrator has control over the minimum interval for these attributes:

- RetestInterval parameter on Wait event actions
- Interval parameter on Sleep event actions
- RetestInterval attributes on event triggers

When any of these attributes is referenced, if the designer-specified value is greater than zero but lower than the minimum interval for the Framework Event Service, the minimum value is used.

The Framework Event Service logs various levels of messages to the Infor framework **Log Monitor**.

Setting Up the Framework Event Service

On the **Event Service** tab of the **Service Configuration Manager**, you must individually specify each configuration that you want to monitor.

Note: If you do not do this setup, any event handler you create that requires the event service will not work.

For more information, including the procedure, see the online help for the Service Configuration Manager utility.

If you want to use the IDO Runtime Development Server (IDORuntimeHost.exe) for development work, you must also temporarily remove the dependency that the Event Service has on the IDO Runtime service. For more information, including the procedure to do this, see the online help for WinStudio developers.

Processing Order in the Framework Event Service

The Framework Event Service processes any queued events in "first in, first out" (FIFO) order.

Because the event service can receive a request to run something while it is executing a prior request, all requests are queued for execution in the order requested. If the new request is only one in the queue and the event service is not busy, the request is executed at the next polling.

When the event queue is empty, and immediately after processing each queued event or event handler, the event service checks the Event Handler State and Event Trigger tables for any items where **Retest At** time setting has arrived or passed.

The event service then checks the Event Handler State table for any items for which the timeout has expired. This is when the **Times Out At** time setting has arrived or passed. The event service processes these in order (oldest first), by using the **Retest At** time or **Times Out At** time to determine the order.

After this, the event service checks the event queue again for any incoming items.

About Event Handler Revisions

The system creates a set of event handler revisions the first time:

- The event is generated or any of its handlers executes.
- The event is generated after any of its constituent handlers or their actions has been modified.
- Any of the event’s handlers executes after any of its constituent actions has been modified.

Modifications can include additions, changes, or deletions to the handlers or actions associated with the event or handler.

When the revision is created, the system copies all of the event's handlers and actions to a read-only table. The event or handler then uses the data in this read-only table when executing the handlers and actions, until a new revision is created.

Revisions are used to ensure that the metadata that is used by an application event and its handlers and actions is not altered while the event is executed. In some cases, it can take the system a period of time to finish executing an event's handlers, either because of the processing time involved, or because the system is waiting for some input or response before it can continue. It is possible that someone could make changes to the event's handlers and actions while the event is processing or waiting, and that these changes could affect the execution of the event that is in progress.

Revisions were developed to address this kind of potential problem. When an event fires, the application event and its handlers use the revisions that are in effect at the time they start executing until they are finished.

Example

You are using an application event to send notices to a manager when a customer order is more than $10,000. The manager must approve the order before it can be processed.

During a transfer of responsibilities from one manager to another, a new manager is assigned the responsibility of approving such orders. The system administrator makes the change of notice to the appropriate event action.

However, the original manager has a few orders pending approval. These orders continue to use the metadata for the revision in effect at the time they were created and await that manager's approval. In the meantime, the new manager receives notices for any subsequent orders, because the first new order generates a new revision using the information for the new manager.

Designing and Using Application Events and Handlers

Infor has built in to the system a number of application events and/or handlers that are available for immediate use. In addition, if you have an add-on product distributed by one of Infor’s business partners, they may also add their own application events and handlers that you can use.

You can also design and create your own custom events and handlers to automate tasks for your particular needs.

Note: If you are creating and using your own custom events and handlers, we recommend that you refresh the metadata cache periodically. This process should be done after doing development work, before testing, and after synchronizing your metadata on your system.

About the Access As Identifier

One key element of the application event system is the Access As identifier. The Access As identifier is used to:

- Indicate who (what organization) created and owns an application event or related event system object
- Prevent unauthorized developers from modifying or deleting event system objects that they do not own

The Access As identifier is also used to indicate ownership and modification rights for certain IDO metadata.

Generally, the Access As identifier falls into one of three classifications:

- `Core` – Indicates that the object is one that Infor created and owns. These event system objects are used for the framework forms and operations.
- *OtherName* – Indicates that the object is created by and belongs to an application created either by Infor or by one of Infor's business partners or other authorized vendors.
- Blank – Indicates that the object was created by and belongs to the end customer.

Within WinStudio, several forms include an **Access As** field. On the **Access As** form, this field indicates the current Access As value. This is the value assigned to any new event system objects you create. On all other forms, this field indicates who has ownership of the pertinent object metadata, in other words, who created and owns it.

You can only modify or delete metadata for event system objects that have the same value as on the **Access As** form, in other words, application event system objects that your organization has created and owns.

With a few exceptions (noted where applicable), you can attach your own event triggers and handlers to event system objects owned by other organizations (that is, with a different Access As identifier than yours), but you cannot directly modify or delete those event system objects.

About Application Events

An application event is defined as a uniquely named situation that can be triggered by:

- An action performed by somebody working in the system
- A condition that occurs while the system is running
- A certain value that is exceeded in a database record
- Another event's handler
- Other similar occurrences

Application events can be one of these general types:

- Framework events – These are events that Infor has defined and built in to the system. They are tagged with an Access As identifier of `Core` and a `Framework` indicator on the **Events** form.

  These events generally fall into one of these categories:

  - IDO (business process-related) events that are generated when certain IDOs are invoked.

These IDOs include IdoOnItemInsert, IdoOnLoadCollection, IdoOnInvoke, and others. You can identify these events easily by their names, which begin with the letters `Ido`.

  - Session events that are generated when certain session activities take place.

    These include SessionOnLogin, SessionOnLogout, and SessionOnVarChanged.
  - Business Object Document (BOD) events that are generated when certain BOD-related operations occur

    You can identify these events easily by their names, which begin with the letters `Bod`.
  - Task events that are generated when certain operations are initiated by the TaskMan service

    You can identify these events easily by their names, which begin with the letters `Task`, excluding TaskListCheck.

  These events are always synchronous and transactional. Some can be optionally suspended to await user responses.

- Application-specific events – These are application events that typically have been created by Infor, its business partners, and authorized vendors. They are tagged with an Access As identifier that indicates what application or development organization they belong to, which can include Core.
- Customer-defined events – These are events that a developer in an end-customer organization has created. They are tagged with a blank Access As identifier, which indicates that they were created by and belong to the customer.

Creating Application Events

Application events can be created or named on these forms:

- Events
- Event Triggers
- Event Handlers

To create your own custom events to use as part of the application event system:

1. Open the **Events** form.
2. Verify that **Filter-in-Place** is off.
3. Select **Actions > New**.
4. Specify this information:

   Event Name
   Specify the name that identifies the application event in the system.

   Description
   Optionally, specify a functional description that identifies the application event's intended use.

5. Save your changes.

Note: You can specify the application event in the **Event Name** field of the **Event Triggers** form and **Event Handlers** form. However, application events that are named on those forms are not displayed on the **Events** form. If you want the application event to display on the **Events** form, you must name the application event by using that form.

Modifying Application Events

You can only modify the application event's description after the application event is created and saved. The application event name and other attributes are locked.

Note: You can only modify an application event's description if the **Access As** field value for the application event is the same as the current value on the **Access As** form.

To modify an event description:

1. Open the **Events** form.
2. Select the application event that you want to modify.
3. In the **Description** field, modify the description text as desired.
4. Save your change.

Deleting Application Events

You can only delete an application event if the event's **Access As** field contains the current Access As value, as displayed on the Access As form.

To delete an application event:

1. Open the **Events** form.
2. Select the application event that you want to delete.
3. Select **Actions > Delete**.
4. Save your changes.

About Event Triggers

An event trigger is a condition that causes an application event to fire, independent of anything that may be happening in the user interface. The event trigger carries a set of event trigger parameters for use when the application event fires.

An event trigger can be set to fire the application event only once, or can be set to retest for its condition after waiting a certain amount of time since either of these situations was true:

- The trigger last successfully fired the application event.
- The trigger last tested unsuccessfully for its condition.

In both cases, you can set the interval for the event trigger to wait, both for the successful firing of the trigger and for the unsuccessful test for the trigger conditions by using separate settings. Testing and retesting is accomplished by means of polling; this is not a true interruptive trigger.

An event trigger carries with it the user name and configuration in effect at the time it was defined. This data is passed on to the event state when the trigger fires the application event.

Each event trigger must contain a condition that consist one of these expressions:

- A Boolean expression

- Two non-Boolean expressions that are separated by a comparison operator

Examples

- This example causes the application event to fire when seven days have elapsed since the current result of the database function dbo.LastEntryDate():

  ```
  DATEDIFF(day, DBFUNCTION("LastEntryDate"), CURDATETIME()) > 7
  ```
- This example causes the application event to fire when the balance on a certain customer's order is greater than $10,000:

  ```
  DBFUNCTION("OrderBalance", GC(BigCustNum)) > 10000
  ```
- This example causes the application event to fire on the first day of each month:

  ```
  DATEPART(day, CURDATETIME()) = 1
  ```

Note: The condition should generally involve a time operation, a database calculation, or both. This is because time and the database are the only known factors that can undergo change from external stimuli (that is, by the forward movement of time or by the actions of other application users, respectively).

Creating Event Triggers

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

- By user actions, such as saving and closing a form, changing a record, and so on
- 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. Select **Actions > Filter > Cancel in Place**.
3. Select **Actions > New** .
4. In the **Event Name** field, select the application event for which you want to define a trigger.

   Note: You cannot define a trigger for a framework event.
5. In the **Trigger > Condition** field, specify the condition for which the application event is to fire.
6. On the **Parameters** tab, specify the names and values for any event parameters for which you must pass values to the event handlers when the event fires.
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.

Retesting Triggers

When the **Event Trigger** > **Retest At** date setting becomes older than the current system clock time, the event trigger is available to be processed by the event service. This happens when the event service is free from processing any waiting queued events and handlers and any already-waiting triggers that need to be tested or retested.

To process the event trigger, the system analyzes and evaluates the condition.

- If the condition evaluates to True, then the application event fires. At that point, the **Retest At** date setting is set to the current time plus the amount of time set for the **Trigger Reset Interval**. If the **Trigger Reset Interval** is set to `0` (zero), then the **Active** check box is cleared, which indicates that the trigger is not to be retested.
- If the condition evaluates to False, then the **Retest At** date setting is set to the current time plus the amount of time set for the **Condition Retest Interval**. If the **Condition Retest Interval** is set to `0` (zero), then the **Active** check box is cleared, which indicates that the trigger is not to be retested.

About Event Handlers

An event handler defines the actions to be taken upon the firing of a particular application event. Each event handler is comprised of one or more event actions and, optionally, an initial state.

Each application event can have multiple event handlers that execute when the application event fires. In such cases, the handler sequence number and other factors determine the order in which event handlers are actually processed.

Creating Event Handlers

In its most basic form, an event handler consists of these parts:

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

Each event handler is uniquely defined in the system by the combination of an event name and a handler sequence number. Both of these are set on the **Event Handlers** form. Event handlers must have one or more associated event actions.

To create an event handler:

1. Open the **Event Handlers** form.
2. Click **Filter-in-Place** or press **F3**.
3. Select **Actions > New**.
4. In the **Event Name** field, select an existing application event or specify a new event name for the handler that you want to define.
5. Specify a description for the handler. This description is helpful when you have multiple handlers with the same event name. This description allows you to find the appropriate handler in a list.

6 Optionally, use the **Keep With** and **Chronology** fields to control the handler sequence with respect to other developers' handlers.
7 Save the new event handler.
8 Click **Event Actions** to open the **Event Actions** form.
9 Create the actions to be performed when this event handler is executed.
10 Save your changes.
11 Close the **Event Actions** form.
12 On the **Event Handlers** form, set the other options as desired. For information about these options, see the online help for each option.
13 To see a graphical (diagrammatic) representation of the event handler and its actions, click **Diagram**. This button opens the **Event Handler Diagram** form. This form allows you to edit and add event actions to the event handler flow.
14 Save the event handler.

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

Using the Event Handler Diagram Form

Note: Event diagramming uses the Nevron Diagram for .NET tool. For more information, including complete documentation, go to www.nevron.com.

You can use the **Event Handler Diagram** form for these tasks:

- 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 you copy, print, or save it.
- Copy the diagram to the Windows clipboard.
- Print the diagram.
- Save the diagram as a graphics file.

About Event Actions

An event action is defined as a unit of work to be performed during the execution of an event handler.

A single event handler can have multiple event actions, but each action is assigned to a single event handler.

Depending on its action type, an event action can do these things:

- Evaluate and compare expressions by using the results to select which event action of its event handler to perform next
- Affect the application event's visual state

- Complete the event handler
- Set event variables
- Call methods or web services
- Perform other predefined tasks

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 **Event Actions**.
4. To create an action for an event handler that already has actions defined, in the grid, create a new row.
5. In the **Event Actions** form, specify this information:

   Action Type
   Select the type of action to be performed. For the list of action types and what they do, see the help for this field.

   Action Description
   Specify a description for the action. This description is helpful when you have multiple actions with the same action type. This description allows you to find the appropriate action in a list.

6. Click **Edit Parameters**. 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.

   Note: If you are familiar with the parameters, functions, and syntax for the action parameters, you can click **Show Details** 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.

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

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

8. 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.
9. Save your changes.

10 Close the form.

Showing Event Action Contexts

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

Example: 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. When you click this button, the new purchase order record is automatically displayed in the **Purchase Orders** form.

Note: This procedure can be performed only for Notify or Prompt actions, 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. On the **Event Actions** form, click **Edit Parameters**.
3. On the **Event Action Notify** or **Event Action Prompt** form, designate the name of a form in the **Filter Form** field.

   This argument designates the form that 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.
4. Click **OK**.
5. On the **Event Action Notify** or **Event Action Prompt** form, designate an expression resolving to a string involving the property on which you want that form to filter when it opens.

   This expression typically takes a form similar to this:

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

   This statement allows the system to pass the specific record that triggered the event handler.
6. Set up the rest of the Notify or Prompt action as desired.

Setting Event Action Variables from a BOD Template

You can use the **Event Action Set Variables From BOD** form to set any number of variables, session variables, and/or parameter values.

1. On a `Set Values` event action type, click **Edit Parameters**.
2. In **Event Action Set Values > Values From BOD**, click **Edit**.
3. In **Event Action Set Variables From BOD > Tree View**, select the node from which the value is to be set.

The Tree View displays the template currently being used. If no template is specified, this view displays whatever is specified for the BODXML value.

The Tree View includes several options to make it easier to work with the template for the inbound BOD:

- Click **Expand All** to display the entire tree hierarchy so that you can see all nodes.
- Click **Collapse All** to display only the top level of the tree, hiding all nodes.

  This allows you to select which nodes you actually want to view more easily.
- Click **Specify Name below to Add/Rename** to specify the name of an element or attribute to either add or rename:
  - To add an element, specify a name here and click **Add Element**.
  - To add an attribute to a node, specify a name here and click **Add Attribute**.
  - To rename an element or attribute, first, in the Tree View, select the element or attribute you want to rename; specify the new name here, and then click **Rename**.
- Click **Add Element** to add an element node to the tree.

  Before you can add the element, you must first select the node under which the node is to be added, and then specify the name of the element to add in the **Specify name below to add/rename** field.
- Click **Add Attribute** to add an attribute node to the tree.

  Before you can add the attribute, you must first select the node under which the node is to be added, and then specify the name of the attribute to add in the **Specify name below to add/rename** field.
- Click **Remove** to remove a selected node from the Tree View.
- Click **Rename** to rename a node selected in the Tree View.

  Before you can rename the node, you must first select it in the Tree View, specify the new name in the **Specify name below to add/rename** field, and then click this.

4. In the **Set Value** section, use the **As** field to specify how the value is to be used:
   - An event **`Variable`**
   - An event **`Parameter`**
   - A **`Session Variable`**
5. In the **Name** field, specify a name to be assigned to the value.

   Optionally, you can specify a default-type value from the drop-down list. For convenience, the application supplies several auto-name options, depending on the type selected in the **As** field: **`Var1`**, **`Var2`**, and **`Var3`** for variables; **`Parm1`**, **`Parm2`**, and **`Parm3`** for parameters; and so on. You are not required to use these suggested names, you can enter your own name in this field.
6. Click **Save**.
7. Repeat these steps to define all the variables and parameters that you require.

The XPath field indicates the XPath of a selected node within the XML hierarchy.

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 application.

1 Open the **Maintain BOD Templates** form.

2 Specify this information:

   Template Name

   Specify a name to identify the template.

   If you leave this field blank, the system automatically uses the name of the XML file that you download.

   Version
   Specify the version number of the template.

   Description
   Optionally, provide a description of the template.

   Active

   Select this check box to make the template available for immediate use.

   If you clear this check box, the template is not listed in the **Templates** field of the **Event Action Send BOD** form.

3 Click **Upload Template**.

4 Browse for the XML file that contains the BOD template.

5 Click **Upload**.

6 Click **Save**.

On the **Templates** tab, the template is loaded and the XML code is displayed.

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

Setting up the Extraction XML Collection Action

Use the Extract XML Collection action to extract a row or a collection of rows from an XML string or from a section of an XML string. Each row in the collection must have the same predefined set of properties.

A collection extracted in this way is equivalent in operation to one resulting from a Load Collection event action. This means that:

- It can be manipulated by a subsequent Update Collection action.
- On any subsequent relevant action, it can be converted back to an XML string using the XML() function.
- On any subsequent relevant action, its rows can be repeated using the ROWS() function, and each row's properties can be referenced using the P() or FP() functions.

This event action provides a convenient way to work with XML data without having to parse through the tags.

To set up the action and its parameters:

1 Create an event action with the action type Extract XML Collection.
2 When you click **Edit Parameters**, the **Event Action Extract XML Collection** form is displayed.
  This form has these buttons, each with an accompanying field:

Button/Field	Notes/Comments
XML Template	The button opens the **Event Action Select XML Template** form, which allows you to select and load the XML template to use with the XML data being extracted. You can also type the XML template in this field manually, or paste it from another source. An XML template is an XML fragment that has the same structure as a single row of data in the XML from which the data is to be extracted. This XML fragment can thus serve as a template for that row of data. The field displays the XML template as a single string.
XML	The button opens an instance of the **Event Action Expression Editor**. The XML referred to here is the XML string from which data is to be extracted. When this type of event action is part of an event handler: • With a BodOnReceive event, a commonly used function is BODXML(), which extracts data from the inbound Business Object Document (BOD). • You can also: • Use the FILECONTENTS() function to read in the contents of a fixed XML file. • Use the XML() function to construct an XML string from another collection. • Reference a variable or parameter. • Manually type a fixed XML string into the field.
Path	The button opens an instance of the **Event Action Set Extract XML Collection** form that allows you to set the extraction path for the XML. This path is to one or more nodes in the XML string under which the data resides for the rows to be extracted. This is expressed in XML Path Language (XPath) syntax.
Property Map	The button opens an instance of the **Event Action Set Extract XML Collection** form that allows you to create one or more sets of property name/value pairs that map values to be extracted from the XML string to properties in each row of the resulting collection. The property map consists of a list of comma-separated pairs. Each pair, in turn, consists of a property name and a path separated by an equals sign. • The property name governs how the extracted value can be referenced later in each row of the extracted collection. • The path describes the location of the data to be extracted for the named property, relative to each location in the XML string identified by the **Path** field. **Note:** This path is expressed in XML Path Language (XPath) syntax.

Button/Field	Notes/Comments
Result Set Assignment	The button opens the **Event Action Set Result Set ID** form, which allows you to provide a name for the set of records that result from the processing of this event action.

3 Set these parameters and click **OK** to close the form.

About Event Action Parameters

Depending on the action type, you specify optional parameters for each event action type. You can specify parameters in any order for a particular action. To list multiple parameters, specify them one after another, specifying either a space or nothing between them. Event action parameters are defined on the **Parameters** tab of the **Event Actions** form.

Note: You can use the event action parameter forms to define the parameters. When you do, the parameters are returned to the **Event Actions** form properly formatted and free of syntax errors.

The syntax for each event action parameter is *FUNCTION(value)*.

Note: Although it is not a requirement that function names be specified by using all uppercase letters, we recommend the practice, as it leads to greater ease of recognition and readability.

The value enclosed in parentheses can consist of:

- A constant number
- A literal string enclosed in quotation marks
- A Boolean value: TRUE or FALSE
- An event function call

  Function call can be nested.
- An expression consisting of a number of these elements that are combined by using operators

You can also use the parentheses after the function to wrap expressions that signify operations to be performed on the results of the expression.

For example, the function *V* takes as a parameter the name of a variable. This function can be placed in the parameters for other functions, for example, *METHOD(V(FuncNameVar))*.

Function types

Functions can be any of three basic types:

- Parameter functions — These are functions whose parentheses wrap a parameter to the event action. For example, these are all typical parameter functions:
  - SETVARVALUES
  - METHOD
  - INTERVAL
  - EVENTNAME

These functions must always appear at the root level and can never be nested inside any other type of function.

These functions are identified in this documentation generically as PARAMS(…).

- Value functions — These are functions that call event values such as:
    - SUBSTITUTE
    - DATE
    - ABS
    - CEILING

  These types of functions can never appear at the root level but must be nested within another function construct (either a parameter or another function).

  These functions are identified in this documentation generically as FUNCTION(…).
- Word functions — These are verbatim words used inside event function calls, such as:
    - AS
    - STRING
    - NUMBER
    - DAY
    - DATE

  Note: Some of these functions can also be used as event function calls. However, these functions always appear within an event function call.

  These functions are identified in this documentation generically as …WORD…

Nesting Function Calls

When defining event action parameters, keep these rules in mind:

- PARAMS(…)-type functions can never be nested.
- FUNCTION(…)-type functions must be nested and can be nested either within PARAMS(…) functions or within other FUNCTION(…) functions.
- …WORD…-type functions must be nested within FUNCTION(…)-type functions.

This is an example of an event action parameter that uses nested functions:

PARAMS(…FUNCTION1(…FUNCTION2(…WORD1…)…WORD2…)…)

Passing Parameters from Actions

Parameter lists to methods, scripts, web services, and generated events are always enclosed in a PARAMS(…) function and delimited by commas. Each parameter is specified in one of these ways:

Syntax	Direction	Meaning
V(*var*)	Input	Pass in the value of the variable *var*.
expression	Input	Pass in the value obtained by evaluating the expres-sion

Syntax	Direction	Meaning
RV(*var*)	Input and output	Pass in the value of the variable *var*, and place the output value into the same variable.

Setting Variable and Parameter Values

You can set values for event variables and parameters by using these syntaxes:

Event action type	Storage type	Syntax
Call Database Method Call IDO Method	Variable	PARMS(RV(*var*))
	Parameter	PARMS(RE(*param*))
Generate Event Load IDO Row	Variable	SET(RV(*var*)=*name*)
	Parameter	SET(RE(*param*)=*name*)
Set Values	Variable	SETVARVALUES(*var=expr*) or adjusted through SETVARENTRIES(..) or SETVARNAMEDENTRIES(...)
	Parameter	SETPARMVALUES(*param=expr*) or adjusted through SETPARMENTRIES(..) or SETPARMNAMEDENTRIES(...)

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.

  To access these forms, select an **Action Type** and then click **Edit Parameters** on the **Event Actions** form.
- Specify 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. You can also begin by directly entering starting parameters in the text edit field on the **Event Actions** form and then adjust the input using the event action parameter forms.

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.

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 any of these types:

- 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 these forms:

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

  For example, the **Condition** button on many event action parameter forms 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 application event system to use.

When you click **OK** in the event action parameter form, the values you specified are returned to the parent form, formatted using the correct syntax.

To verify the syntax is error-free, click the **Check Syntax** button before you proceed.

Example

As part of an application event that notifies a manager when a customer's credit limit has been changed, 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 must perform these steps:

1. On the **Event Handlers** form, create an event handler that runs every time a customer's credit is changed.
2. Click **Event Actions**.
3. On the **Event Actions** form, specify the action sequence number and select the **`Branch`** action type.
4. Click **Edit Parameters**.
5. On the **Event Action Branch** form, click **Condition**.
6. On the **Event Action Parameter Condition** form, click **Expression 1**.
7. On the **Event Action Expression Editor** form, select the **`PROPERTY`** function and specify **`CreditLimit`** as the first argument.

8 Click **OK**.
9 In the **Event Action Parameter Condition > Operator** field, select the **>** (greater than) symbol.
10 In the **Event Action Parameter Condition > Expression 2** field, specify `500000`.
11 Click **OK**.
12 In the **Event Action Branch** > **Destination** field, perform these actions:
   - If the target action sequence step exists, select the number of the action sequence step you want the handler to go to.
   - If the target action sequence step does not exist, specify the number of the step that you plan to create for the target later.

13 Click **OK**.
14 On the **Event Actions** form, click **Check Syntax** to verify that the parameter syntax is all correct.

   If there is an error, an error message is displayed. You can use this error message to determine what and where 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 you click **Edit Parameters** again, you will lose all parameter text and need to start over.

15 Click **Save**.

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 want to specify a group of recipients to receive various notifications and prompt messages, and that group membership changes often. 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 the global constant in one place and all event handlers that use that global constant automatically pick up the change.

Syntax for Expressions

Use quotation marks to indicate a literal string value. For numbers, dates, or Boolean values (TRUE/FALSE), quotation marks are not required or allowed.

Consider these examples:

- `CONDITION( "CreditLimit" < "500000" )`

  Both sides of the comparison are recognized as literal string values and are treated 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 )`

  A syntax error is returned, because **CreditLimit** is not recognized as a valid function.

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

  Both sides of the comparison are valid, 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 the expected result.
- `CONDITION( E(MG_CurrentSite) IN ("MI";"ZZ"))`

  The MG_CurrentSite parameter is set for all running events. It is used 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 )`

  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.

About 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 these functions are available from the **Select a function** field on the **Event Action Expression Editor** form. You can also use them by typing them manually into an expression in the **Parameters** field of the **Event Actions** form.

- ACTIONSEQ
- ACTIONTYPENAME
- ANYHANDLERSFAILED
- APPNAME
- ATTACHMENTEMBEDDED
- ATTACHMENTLIST
- ATTACHMENTNAME
- ATTACHMENTS
- ATTACHMENTSEQ
- BEGINDATE
- BODNOUN
- BODVERB
- BODXML
- CAST
- CEILING
- CLIENTSUBSTITUTE
- COMPANYNAME
- CONFIGNAME

- CURDATETIME
- CUSTOMDELETE
- CUSTOMINSERT
- CUSTOMUPDATE
- DATE
- DATEADD
- DATEDIFF
- DATEPART
- DBFUNCTION
- DOCDESC
- DOCEXT
- DOCINTERNAL
- DOCMEDIATYPE
- DOCMODIFIED
- DOCNAME
- DOCREADONLY
- DOCSEQ
- DOCTYPE
- E
- ENTRY
- ENTRYNAMES
- EVENTNAME
- EVENTPARMID
- EVENTREVISION
- EVENTSTATE
- EVENTSTATEID
- EVENTTITLE
- FE
- FGC
- FILECONTENTS
- FILTER
- FILTERMETHODPARM
- FILTERPROPERTY
- FILTERSTRING
- FILTERTASKPARM
- FLOOR
- FP
- FSV
- FV
- GC
- HANDLERACCESSAS
- HANDLERIGNORESFAILURE
- HANDLERSEQ
- HANDLERSUSPENDS

- HANDLERSYNCHRONOUS
- HANDLERTRANSACTIONAL
- HASBEGUN
- HASFINISHED
- IDO
- IF
- INITIATOR
- INSIDEDATABASE
- INSTR
- LEN
- LOADFLAGS
- LOWER
- MESSAGE
- METHOD
- METHODPARM
- METHODPARMS
- NAMEDENTRY
- NEWGUID
- NONRESPONDERLIST
- NUMENTRIES
- ORIGINATOR
- P
- POSTQUERYACTIONS
- POWER
- PROPERTY
- 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
- TASKNAME
- TASKNUMBER
- TASKPARM

- TASKPARMLIST
- TASKPARMS
- TASKSTATUS
- TRUNC
- UPPER
- USERDESC
- USERNAME
- V
- VARIABLENAME
- VARIABLEVALUE
- VOTINGDISPARITY
- VOTINGRESULT
- VOTINGTIE
- WORKINGDIR
- XML

In addition, use these "pre-parser" functions when you need an expression that contains elements for other functions/expressions:

- TGC
- TV

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 these filter functions:

- 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, a parameter named Prefix is passed to your application event. This application event is designed to do something to all the items whose item ID code begins with the prefix. You can 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 application 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.

About Event Variables and Initial States

If an event handler has variables associated with it, those variables can be assigned an initial state. This set of values are assigned when the event handler starts to execute. Each initial state consists of:

- A name that identifies it in the system
- Any number of event variables with the initial values they are to have when the event handler starts to execute

For example, you want to use these initial values with your event handlers variables:

Name of variable	Initial value	Comments
Increment	250	
ItemTypes	AB	
TimesRemaining	GC(MaxTimes)	In this case, the value is determined by the value of the MaxTimes global constant.

Initial states are defined by using the **Event Variable Groups** form. Once created, you can use a defined initial state with any other event handler by selecting it in the **Initial State** field on the **Event Handlers** form.

About Event Global Constants

An event global constant is a named static value that event expressions can reference during processing of the associated event handlers.

Defining and Using Event Global Constants

Event global constants are defined by using the**Event Global Constants** form.

The system references a global constant by using a function mechanism that allows dynamic evaluation at each reference.

Event global constants are used when defining a set of choices to offer the recipients of a prompt message. For example:

On the **Event Global Constants** form, specify this information:

Name
Specify **`PromptChoicesYesNo`**.

Access As
This field displays the current Access As identifier, which identifies who created the metadata object.

Value
Specify **`1,sYes,0,sNo`**.

You can then reference this constant for any prompt event action by using the expression:

```
CHOICES(GC(PromptChoicesYesNo))
```

Example: Using an Event Global Constant

You have one person in your organization who is authorized to review and set customer credit limits. You can use an event global constant to refer to that person and then use it in a variety of ways. You can set up these events:

- Email this person when an order exceeds a customer's credit limit.
- Notify this person when a customer has been placed on credit hold for some reason.
- Prompt this person for a response to a request for an increase to a customer's credit limit.

Later, if this person is replaced by another person who has the same respnosibilities, you can simply change the reference on this form, so that all these actions involve the new person.

About the Application Event System Design Forms

The system includes a set of specialized forms created to enable you to create and use your own custom application events. With the exception of the **Access As** form and the **System Configuration Parameters** form in this table, these forms are located in the Explorer under **Master Explorer > System > Event System**. The **Access As** and **System Configuration Parameters** forms are located at **Master Explorer > System**.

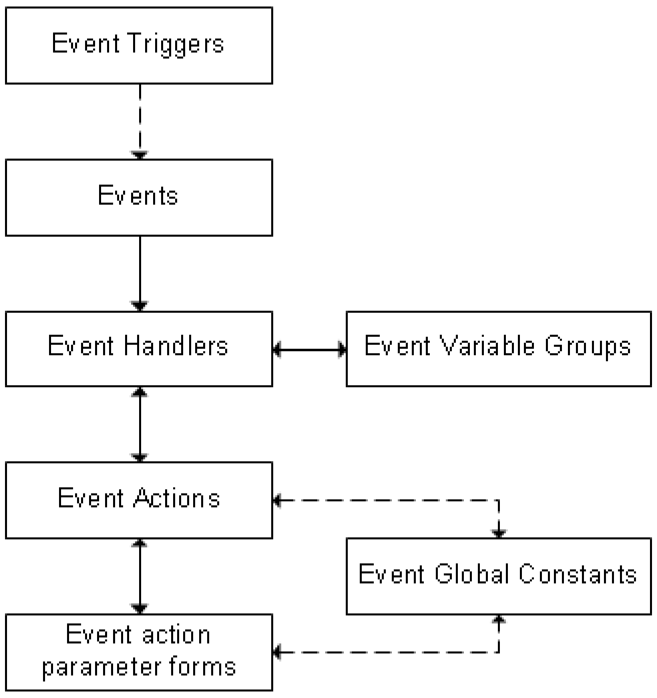
Access to these forms is controlled by the System Administration authorization group.

This table lists and briefly describes the use of the application event system design forms:

Form name	Description
Access As	Although not directly used in the creation and customization of application events and handlers, this form displays the current Access As setting. The Access As value is an indicator of which application event system elements you are authorized to modify and delete.
Events	This form is used to name application events. Once named here, application events are available on other forms as well, particularly the **Event Triggers** and **Event Handlers** forms.
Event Triggers	This form is used to define event triggers, which set conditions that cause a named application event to fire.
Event Handlers	This form is used to display and define event handlers, which determine the work to be done when an application event fires.
Event Handler Diagram	This form is used to present a graphical representation of an event handler flow. You can also use this form to access the **Event Actions** form for selected event actions to view or modify them. Finally, you can also add event actions to an event handler flow by using this form.
Event Handler Sequence	This form is used to change the order of any handlers that have the current Access As identifier as indicated on the **Access As** form.
Event Actions	This form is used to display and define the actions to be performed by a particular event handler during its execution. These are the individual tasks accomplished by the event handlers.
Event action parameter forms	These forms are used to define the event action parameters for each action type.
Event Variable Groups	This form is used to display and define initial states, which are sets of event variables and the initial values they are to pass to the event handler when it starts to run.
Event Global Constants	This form is used to display and define event global constants, which are static values that can be accessed and used by expressions during the running of an event handler.

Form name	Description
System Configuration Parameters	This form is used to define certain parameters that affect the system configuration. Although not directly used in the creation and customization of application events and handlers, you can use this form to control some aspects of how application events and handlers behave on the system. These aspects are concerned mostly with retest and reset intervals that globally govern when and how often various conditions can be retested or events can retry.
Workflow Event Handler Activation	This form is used to activate predefined event handlers that represent common workflows. You must specify some information, such as users or email addresses that are notified when an application event occurs. You can also copy and modify these event handlers. **Note:** This form is not available with all Mongoose-based applications, including Mongoose as a stand-alone application. See the online help in your application for this form.

This diagram shows the functional relationship between the design forms and elements for the application event system:



Setting up Custom Application Events and Handlers

To create and use your own custom application events, there are several important steps and considerations to keep in mind. You must:

- Design and define your custom event.

- Carefully plan and set up the order in which event handlers and actions execute.
- Periodically refresh the metadata cache to ensure that you are working with the most current version of the event metadata.

Designing a Custom Application Event

When you create a custom application event, you must also define what fires the application event. In the current WinStudio system, there are ways to generate a custom event:

- Create an event trigger by using the **Event Trigger** form. This step is the easiest and most common way to generate a custom application event.
- Use **`Generate Event`** action type in another event handler.
- Use a form or WinStudio event handler with **`Generate Application Event`** response type. **Note:** Do not confuse the form or WinStudio event handlers with the application event system handlers.
- Use the WinStudio API to write a custom script that generates an application event.
- Invoke an IDO Extension Class method that calls the FireApplicationEvent() method.
- Use the Mongoose Representational State Transfer (REST) service, **`IDORequestService/MGRestService.svc`**.
- Use a SQL stored procedure or trigger to call the PostEventSp stored procedure.

These steps represent a typical process for creating custom application events:

1 Optionally, on the **Events** form, specify the name of the application event before you define how it should be triggered or handled.

Note: If you do not name the application event on the **Events** form, you can specify the application event name on the **Event Triggers** form or the **Event Handlers** form. However, application events that are named on those forms are not displayed on the **Events** form.

2 On the **Event Triggers** form, define one or more triggers that fire the application event.

Note: This step is required only if you want to fire the application event by using an event trigger.

3 On the **Event Handlers** form, define one or more event handlers that execute when the application event fires.

Note: Each application event can have multiple handlers that execute when the application event fires. The order in which multiple handlers execute is controlled by a number of factors.

4 On the **Event Actions** form, define one or more event actions for each event handler.

5 If required, on the **Event Variable Groups** form, name and define an initial state for the event handler to use.

6 If required, on the **Event Global Constants** form, name and define any global constants for the event handler to use.

7 Test the application event and its triggers and handlers on a test system before implementing them on your live system.

Ordering Event Handlers

When an event handler is first defined and saved, the system automatically assigns a handler sequence number. When the event handler is first saved, the system checks to see if there are other handlers associated with the named application event. Depending on the results of that check, the system then assigns 1 to the handler (if there are no other handlers associated with the event) or the next available integer (if there are other handlers associated with the event).

In general, then, if an application event uses multiple event handlers, by default, the system uses the handler sequence numbers to determine the order in which the handlers execute.

However, It is possible to indirectly alter this default order. This is done either by:

- Using the **Keep With** and **Chronology** fields on the **Event Handlers** form
- Moving your own adjacent event handlers up or down in the sequence by using the **Event Handler Sequence** form

To alter the default order in which event handlers execute using the **Keep With** and **Chronology** fields on the **Event Handlers** form:

1 Optionally, from the **Keep With** drop-down list, select the event handler that you want to use as a reference point and anchor for the current handler.

   Note: This step is not required if you are using the `First` or `Last` option in the **Chronology** field.

2 From the **Chronology** drop-down list, select the option that you want the current handler to use with respect to the handler you selected in Step 1.

   This table shows the list of options:

Option	Description
`First`	Executes the current handler before any other handlers
`Before`	Executes the current handler just before the referenced handler
`Instead`	Executes the current handler in place of the referenced handler. In this case, the referenced (original) handler does not execute at all
`Exclusively Instead`	Executes the current handler instead of the referenced handler and any other handler that may be referenced to execute instead of (`Instead`) that same handler
`After`	Executes the current handler after the referenced handler
`Last`	Executes the current handler after all other handlers

Resequencing Event Handlers

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

Note:

- This form is intended to be accessed as a linked form, only from the **Event Handlers** form through 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 that is 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 your changes.

Ordering Event Actions

A single event handler can contain multiple event actions. When you define an event action, you must assign an action sequence number in the **Action Sequence** field of the **Event Actions** form. You can assign any number you want in that field and the system automatically sorts the actions in the correct sequence on the **Event Handlers** form. When the event handler executes, the system uses this **Action Sequence** number to determine the order in which the actions execute.

The only exception to this rule is that, if you select a particular action (other than `1`) in the **Initial Action** field of the **Event Handlers** form, then processing of the event actions begins with the designated action and proceeds from there. So, for example, if you have four actions associated with a handler, and you later decide that you want action number 3 to be the starting point, you would select 3 in the **Initial Action** field. In this case, only actions 3 and 4 executes. The system skips over actions 1 and 2. You can later execute these actions by using **`Branch`** or **`Goto`** action types, with one of these actions as the destination.

Determining Names of IDO Collections and Properties

To create custom event handlers and actions, you are often required to know the internal names of the collections, known internally as IDO collections, and the properties you want to refer to. For example, to set up a handler, you often need the name of the IDO collection associated with a particular form. To include dynamic content in the subject or body of a message, you often must know the internal name of a property within that IDO.

Determining the Name of an IDO Object

To determine the name of an IDO object for an event handler definition:

1. Open the form that uses the IDO collection you require.

For example, if you are setting up an event handler to work with customer records, open the **Customers** form.

2. Go into Design Mode for that form.
3. Verify that the **Form** properties sheet is displayed.

   If not, select the **View** → **Form Properties** option.
4. Select the **Collections** tab.

   The names of all IDO collections associated with that form are displayed in the Collections list at the top of the form. Usually, there is only collection, which makes it easy to figure out. If more than one collection is listed, you must take further steps to determine which is the one you need. properties sheet is displayed.

   The internal name of the IDO is that part of the IDO name that displays after the period. For example, the **Customers** form uses the SL.SLCustomers IDO. The name of the IDO collection as you need it to create an event handler, is **`SLCustomers`**.

Determining the Name of a Property bound to a Form Component

To determine the name of a property bound to a form component:

1. Open the form that has the field or other component that displays the data to which you want to refer.

   For example, to create a subject line that includes a customer ID number and name, open the **Customers** form.
2. Go into Design Mode for that form.
3. Select the desired component, for example, the **Customer ID** field.
4. Verify that the **Component** properties sheet is displayed.

   If not, select the **View** → **Component Properties** option.
5. On the **Component** properties sheet, locate the **Data Source** → **Binding** field.

   The component's property name is displayed in that field after the period. For example, for the customer ID number, the **Binding** field displays **`object.CustNum`**. Thus, the **Customer ID** field is bound to the property named **`CustNum`**.

Refreshing the Metadata Cache

Because certain event metadata is cached for faster performance, the IDO metadata cache must be refreshed periodically: after changes to event metadata (that is, after making changes to events, handlers, actions, triggers, and global constants). This must be done, at a minimum, after doing development work, before testing, and after synchronizing the metadata on your system.

Note:
- If you have multiple application servers in your system, you must refresh cached metadata for each application server on which metadata might have been cached. The best way to do this is by unloading global objects from your system.

- Any event metadata that is not referenced within two minutes is automatically refreshed from the cache. That is why you might notice that things work the way you expect, even without manually refreshing the cached metadata. We recommend that, as a precaution, you manually refresh the cached metadata to be sure.

These are the ways you can refresh the cached metadata:

- Click the **Discard Cache** button on the **Utilities** tab of the Configuration Manager.
- Unload global objects.

  This requires that you first select the **Unload IDO Metadata with Forms** option in **User Preferences**.
- Restart the IDO Runtime service on the Applications server or the local instance of the IDO Runtime host service.

Refreshing Cached Metadata by Unloading Global Objects

With the correct system settings, you can have the system refresh the cached metadata automatically every time you unload all global objects.

To refresh cached metadata by unloading global objects:

1. In WinStudio, select **View > User Preferences**.
2. Select the **Runtime Behaviors** tab.
3. Select the **Unload IDO Metadata With Forms** check box.

While working on event metadata, periodically unload global objects, and the cached metadata is automatically discarded and refreshed with the global objects.

Refreshing Cached Metadata by Restarting the IDO Runtime Service

If you do not want to refresh the cached metadata by unloading global objects from your system and you are not using the IDO Runtime Development Server on your local machine, you can discard the cached metadata by manually stopping and restarting the IDO Runtime Service on the application server. This works because the metadata cache is not saved when the service is stopped.

To refresh cached metadata by restarting the IDO Runtime Service:

1. On the application server, open **Control Panel**.
2. Select **Administrative Tools > Services**.
3. From the list, double click **Infor Framework IDO Runtime Service**.
4. In the **Infor Framework IDO Runtime Services Properties** dialog box, click **Stop**.
5. When the **IDO Runtime Services Properties** dialog box indicates that the service has stopped, click **Start**.

Refreshing Cached Metadata in the IDO Runtime Development Server

If you are using a local installation of the IDO Runtime Development Server, you can refresh the cached metadata manually.

To manually refresh cached metadata in the IDO Runtime Development Server:

1 In the IDO Runtime Development Server, select the configuration for which you want to refresh the cached IDO and event metadata.
2 From the **Configuration** menu, select **Discard IDO Metadata Cache**.

Tracking Application Event System Status

Some application events can take a considerable amount of time to process, especially if they involve event messages that require responses from the recipients. The system provides a number of tools and forms that allow you to track the status of application events as they execute and after they have finished executing. Some of these forms also allow you to temporarily adjust the behavior of handler execution.

These forms are located in the Explorer under **Master Explorer > System > Event System**.

Initially, these forms can be accessed only by members of the System Administration authorization group.

These forms can all be used to track various aspects of event system status:

- Event Status form
- Event Handler Status form
- Event Queue form
- Event Revisions form
- Event Handler Revisions form
- Suspended Updates form

About the Application Event System Messages

Event messages in the application event system can be generated by:

- The system, as part of an event handler’s actions
  Note: Only Notify and Prompt action types can generate messages.
- Other users on the system, through the **Send Message** form

Each message is visible only by the recipients and optionally, the sender of that message.

Event message-related forms

Event message-related forms are used to view, sort, file, respond to, and send messages generated within the application event system. These message forms reside in the **Master Explorer > System > Messages** folder.

The application event system uses these message-related forms:

- **Inbox**
- **Saved Messages**
- **Send Message**

Responding to System-Generated Messages

If the message is the result of a system-generated prompt, each recipient can respond to the prompt, usually by means of a set of voting buttons.

If the message is also sent to the recipient's external email inbox and the recipient responded to the email message, then the message in the refreshed **Inbox** is marked as expired and the buttons are inactive, so the recipient cannot respond twice.

If the message is system-generated and involves variables, for each variable, depending on the **Variable Access** setting for the event variable (on the **Event Actions** form) or initial state (on the **Event Variable Groups** form) or payload status (on the **Event Action Notify** or **Event Action Prompt** form), recipients can:

- Provide an optional response
- Provide a mandatory response
- Only read the variable value
- Not see the variable value at all

Setting Variable Access

The effective visibility and writability of each variable displayed on the **Inbox** form is determined by:

- What type of action generated the message (Notify or Prompt)
- The (optional) variable access level as specified on the **Event Action** form for the action itself and/or on the **Event Variable Groups** form for the event handler's initial state and/or on the **Event Action Notify** or **Event Action Prompt** form for the action with regard to the payload status of the property that corresponds to the variable, when the handler is associated with an IDO event.

For Notify type messages, the default for variable access is **`Read-Only`**. You can use the variable access options to override this to **`Hidden`** for each variable.

For Prompt type messages, the default for variable access is **`Writable`**. You can use the variable access options to override this to **`Hidden`**, **`Read-Only`**, or **`Mandatory`** for each variable.

To change the value of a variable that may appear with a message in the **Inbox** form, you can do any of these actions:

- Specify the data by using the **Variables** tab on the **Inbox** form as part of a response to a Prompt action.
- Use a Set Values event action with the function syntax: **`SETVARVALUES(VariableName=expression)`**.
- Use any of various event actions that can set a variable on output by using the function syntax: **`RV(VariableName)`**.

Setting Translatable Captions for Variables

On the **Inbox** form, the **Variables > Caption** column displays the contents of the notify or prompt message's variable captions in the current user’s language. This assumes that:

- The **Caption** component attributes are set to interpret the bound contents.
- The **Caption** contains a translatable string name.

For payload variables resulting from an IDO event, this string name may come from an IDO property’s label string ID. For non-payload variables that are created by an event action, define this string name by using the **Variable Access** tab on the **Event Actions** 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** field, select the folder in which the message you want to move is currently placed. All messages in that folder are displayed in the grid.
3. Select the message that you want to move.
4. In the **Folder Name** field, perform one of these actions:
    - To move the message to an existing folder, specify the name of the folder.
    - To create a new folder and move the message to that folder, specify the name of the new folder.
5. Save your change.

Sending Email to External Email Inbox for Prompts

If a user is set up in the **Users** form to allow the system to send external prompts, then a prompt action sends the message both to the application's **Inbox** form and to the recipient’s Inbox in an external email client such as Microsoft Outlook. The message that is sent to the external email system is an HTML-formatted email that consists of these parts:

- Original subject in **Subject** line
- **Category**
- List of internal recipients
- List of internal **Cc** recipients
- Original **Message** (either in plain text or HTML format)
- Original **Question**
- **Choices**, as individual hyperlinks to a .NET active server page (ASP) that records the vote
- Payload, which is the contents of the **Variables** grid from the **Inbox**

Text in the **Subject**, **Category**, **Message**, **Question**, **Choices**, and payload captions can be translated and formatted based on the default language specified for the recipient user in the **Users** form.

When the recipients vote by clicking the link in the email, their Windows-default web browser opens or opens a new tab, as applicable, and sends the information from the link they clicked to the ASP URL that is included (and hidden) in the email. This URL is built by the system based on the web server list.

The ASP then registers the vote programmatically, as if the recipient had logged into the system, displayed the same message in the **Inbox**, and selected the corresponding **Choice** button on the **Response** tab.

The web page then displays a success or failure message.

The message displays both in email and in the **Inbox** form, but only one response is allowed:

- If users respond from email first, when they display the **Inbox** (properly refreshed), the message is expired and the buttons are inactive.
- If users respond from the **Inbox** first, then clicking a link in the external email brings up the web page with a message that the message has expired and they already voted, using the previously selected choice.

About Prompts and Responses

If a message is the result of a Prompt action, the sender of that message can request a specific response from each recipient, usually in the form of a button-vote mechanism. In such cases, the system must be set first to wait for responses, then to know how to handle responses as they are received, and finally, be instructed what to do when responses are not received within a specified timeout period.

Incoming prompts request a response from recipients (using the **Question** field on the **Inbox** form or the external email) and display a set of choices. The choices are displayed in the form of voting buttons in the **Response** tab area of the **Inbox** form, or in the form of links in an external email. For example, a prompt may include buttons or links labeled:

- Approve or Disapprove (default option)
- Yes or No
- OK, Send More Info, or Cancel

To customize the choices for a prompt, you must include a Prompt action with a **Choices** parameter as part of the event action definition. This **Choices** parameters consists of the CHOICES function followed by a string expression that evaluates to a comma-separated list that contains an even number of elements (value/label pairs). For example, if you want the voting buttons to be labeled Yes and No, with corresponding values returned to the action to be 1 and 0, you can include this parameter:

```
CHOICES("1,sYes,0,sNo")
```

In this example, the strings `sYes` and `sNo` are WinStudio form strings. These are defined for the system as Yes and No, respectively.

If you want your button labels to be localized, for the recipient, you must:

- Use names of existing form strings (as found in the Strings table).

OR

- Add your own form strings, using the **Strings** form, and provide the necessary translations.

  To open the **Strings** form, you must be in Design Mode and from the **Edit** menu, select **Strings**.

If localization is not an issue, you can also use a literal value that displays on the button verbatim. To specify the string as a literal value here, simply specify it as a list value. If the system does not find the string in the Strings table, the system automatically treats it as a literal value.

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, you might need to gather more specific or detailed responses. 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.

   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 **`set values`** type 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 **`Row.CoNum`** event variable, 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 (**`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 Click the **Edit Parameters** on the **Event Actions** form to open the **Event Action Prompt** form.

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

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

Note: The **Send External Prompts** check box might be selected for some users on the **Users** form. In this case, you must include a note in the body of the prompt email, instructing users to log in to this application to enter the requested data or perform the actions needed on the custom form.

About Voting Rules in Prompts

When a prompt is sent to a single recipient, the result of the prompt is the return value from that recipient's choice. However, when a prompt is sent to multiple recipients, you must select a vote-counting method to determine the result of the prompt and include a Voting Rule (VOTINGRULE) parameter in your event action definition.

The following table lists and describes the available voting rules.

Rule	Description
Majority	A choice must receive more than 50% of the vote to win. As soon as more than 50% of the recipients respond with a particular choice, that choice wins. For example, suppose a prompt went to nine recipients. Of the first six to respond, five vote in favor. At that point, the vote is over with this option, because five votes is more than 50%. The event handler moves on, and it does not matter what the remaining three recipients do. If you use this voting rule, you should use a Voting Tie (VOTINGTIE) parameter to tell the system how to handle a tied vote.

Rule	Description
Plurality	The choice with the highest number of votes wins, even if the choice does not receive more than 50% of the vote. For example, suppose three choices are offered, and: • The first choice receives 24% of the vote. • The second choice receives 43% of the vote. • The third choice receives the remaining 33% of the vote. In this case, the second choice wins, even though the votes received are less than 50% of the total vote. If you use this voting rule, you should use a Voting Tie (VOTINGTIE) parameter to tell the system how to handle a tied winning vote.
Conditional Plurality	The choice with the highest number of votes wins, but only if a specified minimum percentage of votes is reached. If you use this rule, you must also include a Minimum Percentage (MINIMUM) parameter. For example, if three choices are offered to 19 recipients, and you specify a minimum percentage of 40% to win, then: • In an 8-7-4 split, the choice with eight votes wins because this choice meets the minimum percentage. • In a 7-6-6 split, there are no winners, because no choice meets the minimum percentage. In this case, the system must deal with the vote as an indeterminate result. With a simple Plurality vote, the choice that reaches seven votes in a 7-6-6 split wins. If you use this voting rule, you should use a Voting Tie (VOTINGTIE) parameter or Voting Disparity (VOTINGDISPARITY) parameter to tell the system how to handle the vote.

Rule	Description
Minimum Count	The first choice to reach a specified minimum number of votes wins. If you use this rule, you must also include a Minimum Count (MINIMUM) parameter. For example, if three choices are offered to 13 recipients, and you specify a minimum of five votes to win, the first choice to receive five votes automatically wins. **Note:** As soon as the minimum count is reached, event handler execution moves immediately to the next action. In this case, the system expires any responses not yet received, and no further voting can take place.
Minimum Percentage	The first choice to receive a specified percentage of the vote wins. The percentage is based on the number of recipients of the prompt, not the number of respondents. If you use this rule, you must also include a Minimum Percentage (MINIMUM) parameter. **Note:** As soon as the minimum percentage is reached for a choice, event handler execution moves immediately to the next action. In this case, the system expires any responses not yet received, and no further voting can take place.
Earliest Response	The first response to the prompt wins, regardless of the choice. **Note:** As soon as the first response is received, event handler execution moves immediately to the next action. In this case, the system expires any responses not yet received, and no further voting can take place.

Rule	Description
Preferred Choice	If any one respondent votes for the preferred choice, that choice wins. In a case where none of the respondents select the preferred choice, then this rule behaves as the Plurality rule for the remaining choices. If you use this rule, you should also include a Preferred Choice (PREFCHOICE) parameter to specify which choice is the preferred choice. For example, if you have three choices, and you specify the first choice as the preferred choice, then: • If anyone votes for the first choice, that choice wins. • If the end vote is a 0-6-5 split, the second choice wins. **Note:** As soon as the preferred choice receives a vote, event handler execution moves immediately to the next action. In this case, the system expires any responses not yet received, and no further voting can take place.
Minimum Count Preferred Choice	If a specified number of votes for a specified choice is cast, that choice wins. If you use this rule, you must also include a Minimum (MINIMUM) parameter to specify the minimum count, and a Preferred Choice (PREFCHOICE) parameter to specify which is the preferred choice. For example, if you set the Minimum to **`3`** for a Preferred Choice of **`Approve`**, and three recipients respond with **`Approve`**, the preferred choice wins. If less than that number of votes are cast for that choice after all recipients have responded, the vote reverts to Plurality. In that case, the preferred choice may win. **Note:** When you set the Minimum to `1`, this rule behaves as a Preferred Choice.

Rule	Description
Minimum Percentage Preferred Choice	If a specified percentage of votes for a specified choice is cast, that choice wins. If you use this rule, you must also include a Minimum (MINIMUM) parameter to specify the minimum percentage, and a Preferred Choice (PREFCHOICE) parameter to specify which is the preferred choice. For example, if you set the Minimum to 25% for a Preferred Choice of **`Approve`**, and two of eight of recipients respond with **`Approve`**, the preferred choice wins. If less than that percentage of votes are cast for that choice after all recipients have responded, the vote reverts to Plurality. In that case, the preferred choice may win.

About Voting Status

In the Application Event System, multiple voters can be included in a Prompt action. You can use the **Voting** tab on the **Event Status** form and **Event Handler Status** form to monitor the voting status of a prompt. As an administrator, you can use voting status to determine why a suspended row may not be released yet. The **Voting** tab provides this information:

- Which message recipients have yet to vote
- The number of votes still needed to reach a quorum
- The length of time the workflow has been waiting since voting began, and since the last vote
- Current vote tally statistics for each choice, including:
  - Front-runner: indicates whether this choice is in the lead to win
  - Tied: indicates whether this choice is tied with another choice
  - Margin of Victory/Lead: numerically indicates how far ahead this choice is in relation to second place, or how far behind a choice is in relation to first place
  - Votes to Win: indicates the number of additional votes required to win
  - Status: indicates whether a choice is a contender, the winner, or a loser

Dealing with Indeterminate Voting Results

These example situations create indeterminate voting results and set action attributes that are exposed as event functions that can be evaluated by subsequent event actions:

- Any disagreement among multiple recipients that is registered as soon as a disagreement is detected. This can include a vote similar to the example offered in the Plurality description.

  The associated event function is the VOTINGDISPARITY( ) event function, which is a Boolean function that indicates only that there was a disagreement.

- A tie in the case of a Plurality or Majority vote that is registered at the point when all responses have been received or when the timeout period has expired.

  The associated event function is the VOTINGTIE( ) event function, which is a Boolean function that indicates only that there was a tie.

You can use the returns from these functions, along with the functions RECIPIENTS( ), RESPONDERS( ), RECIPIENTLIST( ), RESPONDERLIST( ), and NONRESPONDERLIST() to take further actions, such as:

- Reprompting all the recipients and try to get a consensus
- Reprompting only a select group of the respondents and urging them to adopt a different choice
- Reprompting only recipients who have not yet responded
- Taking some other predetermined action

About Quorums

On a Prompt action, a quorum is automatically calculated based on the number of recipients, the voting rule, and voting parameters such as **Minimum**. If there is a number of votes by whose tally a voting result can be determined unambiguously, that number is the quorum. Otherwise, the quorum is the number of recipients, that is, everyone has a chance to vote unless a timeout expires. As soon as the quorum is reached, voting is closed, any remaining unvoted messages are expired, and the application event continues to the subsequent event action.

However, if you specify a **Quorum** value, that overrides the automatic calculation. For example, if a message requiring a response is sent to 10 people, but you want a quorum to be reached when only four have voted, then specify `4` as the **Quorum** value.

By default, if **Quorum** is not specified or is specified with a positive value, **Wait for Quorum** is true; that is, the application event waits until the quorum is reached before it continues with the next event action. If **Quorum** is specified with a non-positive value, the **Wait for Quorum** default value is FALSE. If these settings conflict, for example **Quorum** is `3` and **Wait for Quorum** is FALSE, the system displays an error message.

If **Wait for Quorum** is FALSE, the application event does not wait for a quorum to be reached. As soon as the messages are sent, execution continues with the next event action. If the system is not waiting for a quorum, the event designer must determine when and under what circumstances enough votes have been received and exactly what further actions the system is to take. This can be done using VOTINGRESULT(), RESPONDERS(), RECIPIENTS(), and so on, in combination with the Wait or Sleep actions.

Chapter 5: Schema (SQL Table and Column) Editing

Maintaining Tables and Other SQL Schema Elements

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

See these topics to maintain tables and other SQL schema elements from the application:

- Creating Tables on page 110
- Maintaining Columns on Tables on page 111
- Specifying Primary Keys and Other Constraints for a Table on page 111
- Updating Existing Tables on page 112
- Editing SQL User-Defined Data Types on page 112
- Executing SQL Statements on page 113

Creating SQL 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.

   See Specifying Primary Keys and Other Constraints for a Table on page 111.
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: This application 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 SQL Tables

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

See Restricted Tables on page 113.

Caution: You can add, but do not delete or modify columns on existing base application tables. Doing so can cause system instability.

To add a new column to a table that already contains columns, the new column must meet at least one of these conditions:

- Allows nulls
- Has a default definition specified
- Is an identity or timestamp data type

If none of these conditions is true, then the column can be added only to an empty table.

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 this 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 are displayed.

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 SQL Tables

Use this method to update a table that you imported into the application database (that is, a table that was not created with the **SQL Tables** form).

To prepare an existing table for application access:

1. Select the table on the **SQL Tables** form.
2. Click **Update Current Table**.

   These actions are performed on the table:

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

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.
4. Optionally, 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. Perform these steps:

   a. Filter for the data type and verify that the **Where Type Used** grid is empty.
   b. Select **Actions > Delete** and save the record to drop the data type.
   c. Select **Actions > New**, specify the information again, and save the data type record to recreate it.

Executing SQL Statements

1. From the **SQL Tables** form, click **Apply Database** to display a form where you can execute any SQL statement into the application database.
2. 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.

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

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. Asterisks denote that 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

SQL Reserved Words

The application reserves the following words that cannot be used aa 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

Chapter 6: 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.

See Finding the Primary Collection for a Form.

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.

To add new user fields to forms, see Extending Application Database Tables on page 124.

Reports

These 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

To add custom user fields to application forms by extending application database tables:

1. Create a user class.

   The user class definition is the highest level to extend an application database table.

   See Creating a User Class on page 126.

2. Create the 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.

   See Creating User Fields on page 126.

3. Associate the user fields with the user class.

   The UET tools look for this association to place the user fields in the form that belongs to the user class.

   See Associating User Fields with a User Class on page 124.

4. Define the index for the 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.

   See Defining an Index for a Class on page 127.

5. Create a relationship between an application database table and 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.

   See Creating a Relationship Between a Database Table and a User Class on page 125.

6. Impact 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.

   See Impacting the Schema on page 128.

7. Draw the user fields on 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.

See Drawing UET Fields on Forms on page 127.

Copying a 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 Fields** 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

1. Determine the primary table name for the form:
   a. Open the form to which you want to add a UET field.
   b. Select **Edit > Design Mode**.
   c. If the form properties sheet is not open, select **View > Form Properties**.
   d. Select the **Collections** tab on the property sheet.
   e. In the collections tree, select the primary collection.
   f. The **Base Table and Alias** property shows the primary table name.

   Note: Certain tables cannot be extended. See User Extended Tables Overview on page 123.
2. Associate a user class with the application database table:
   a. On the **UET Table/Class Relationships** form, select **Actions > New**.
   b. In the **Table Name** field, select the table, as determined above, with which you want to associate a class.
   c. In the **Class Name** field, select a class to associate with the table.
   d. Select **Active** to be able to draw user fields on application forms.
   e. Click **Rule Assistant**.
   f. Select a field, an operator and a value for the rule.
   g. 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.

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

i Click **OK**.

3 Save your changes.

Creating a User Class

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

2 Specify a **Class Name**, **Label**, and **Description** for the class.

3 Save the class.

Creating User Fields

To create a user field:

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

2 Specify this information:

User Field Name
Specify a unique name, prefixed with `uf`, to define a user field.

The uf prefix is not case sensitive, so you can specify something like `Uf_Category` , `uf_Category`, or `UF_Category` as a user field name.

User Data Type
Select a user data type to define the new user field.

Data Type
Select a standard SQL Server data type for the user field.

Precision
Specify the number of alphanumeric characters that can be entered in the user field.

Decimals
If you selected `Decimal` in the **Data Type** field, enter the number of decimal places available in the user field.

Initial Value
Specify the initial value you wish to display in the user field.

Description
Specify a description of the user field.

3 Save the user field.

About User Defined Types

User-defined type refers to a named group of data values that can be used as a list source for a component. For example, a user-defined type named "Size" could invoke a list of values consisting of "Small," "Medium," and "Large."

You create user-defined types and add values to them in the **User Defined Types** form. **Value** and **Description** pairs display in a two-column list for each type that you define. The list of all values for all user-defined types are included in the **User Defined Type Values** form.

After a user-defined type is specified as the source for a component that supports lists (a combo box, for example), users can add values to the type directly through the right-click Add function. In the example above, a user might add the value "X-Large."

User-defined types are useful with user-extended tables. They can also be used with user-defined fields. Users can add list-type components to forms without modifying the schema or IDOs or writing special form code.

Defining an Index for a Class

To define an index for a class:

1. On the **UET Class/Index Relationships** form, select **Actions > New**.
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* contains a chapter on "Setting Up Custom Replication." In it you can find information about copying table schema changes to other sites.

Chapter 7: Critical Numbers

About Critical Numbers

Critical numbers are key performance indicators, or KPIs, that you can use to get a quick view of important statistics or track progress. Use critical numbers to answer questions like these:

- How am I doing?
- What should I be doing?
- Where are things in jeopardy?

Critical numbers are based on stored procedure calculations or on IDO calculations. With the appropriate permissions, 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.

For example, within the IDO critical number definitions, you can quickly generate numbers based on this information:

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

You can see the details behind the critical number using an "enhanced drilldown" and do further analysis on issues you discover.

Where Critical Numbers Can Be Used

More than 100 critical numbers are predefined for your use. You can also create your own critical numbers.

Critical numbers can be added to any form; the Home forms have several predefined examples. To add a critical number as a component on a form, you must have access to the form (MGDataViews license module) and design mode editing permission to add the component.

Critical numbers can also be added as widgets on a user's **Start** form.

Critical numbers can be surfaced as Infor Ming.le widgets, or used as widgets in the Infor Ming.le HomePages. Use the **User Critical Number Selection** form to configure this.

Critical Number Definitions

Before they can be used to display data on a form, critical numbers must be defined. Critical numbers are defined using the **Critical Numbers Setup** form. On that form, the basic process is to:

- Specify a source for the data that is to be used.
- Define how the critical number's actual value is to be calculated.
- Determine whether you want to use Goal and Alert settings, and if so, how.

  See About Critical Number Goals and Alerts on page 130.
- Optionally, you can also include this information:
  - Specify what critical number drilldowns are to be used when the user seeks more information about how the critical number was calculated and what it means.

    See About Critical Number Drilldowns on page 142.
  - Determine what groups and users will have access to the critical number data, using critical number categories and individual user or group assignments.

    Viewing a critical number is controlled by the permissions set for the specific number. User or group permissions can be set for a category of critical numbers on the Critical Number Category form, or user permissions can be set for a specific number on the Critical Number Setup form.
  - Define critical number parameters and how they will be used in critical number calculations and displays.

    See Setting Up a Critical Number Drilldown on page 142.
  - Determine and set up filters to be used in retrieving critical number data.

    See Setting Up a Critical Number/Drilldown IDO Filter on page 149.
  - Specify whether "snapshots" of critical number values should be captured whenever the critical number is queried for.

    See About Critical Number Snapshots on page 140.

For the complete procedure to define a critical number, see Defining a Critical Number on page 131.

Goal and Alert Settings

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

- **`Red`**: The value of the number is in the alert or critical range.
- **`Yellow`**: The value of the number is in the warning range.
- **`Green`**: The value of the number is within the normal or ideal range.
- **`Gray`**: The number represents an "informational" value only.

You can change these colors. Use themes to do this.

About Critical Number Goals and Alerts

Goals and alerts are used in Critical Numbers to determine how the critical number displays are presented and interpreted. Goal and alert settings indicate thresholds that can be used to indicate when goals are being met, when goals are in jeopardy, and when goals are not being met.

- **Goals**: Anything in the "Goal" range is considered good and by default is displayed using a green color.
- **Alerts**: Anything in the "Alert range is considered bad and by default is displayed using a red color.

- **Warnings**: When both Goal and Alert settings have been defined, a "warning" range is generated automatically between them. This range by default is displayed using a yellow color.

  If you choose to use a Goal setting without a corresponding Alert setting, anything not in the acceptable (goal) range is considered a Warning.

  If you choose to use an Alert setting without a corresponding Goal setting, anything not in the unacceptable (alert) range is considered a Warning.
- **Informational displays**: When neither a Goal nor an Alert setting has been defined, the Critical Number display is considered to be informational only. Informational numbers are values that you want to monitor but not set thresholds against. By default, informational displays use a gray color.

You can change the default colors for each of these types of critical number values, by using themes.

When using email generation with Critical Numbers, you can choose which of these four states (Goal, Alert, Warning, Informational) is to generate emails to appropriate individuals, using the **Email Generation** form.

Defining a Critical Number

Critical numbers help users track how they are doing. Critical numbers can be based on a stored procedure calculation or on an IDO calculation.

Custom critical numbers inherit the same security restrictions as the preconfigured numbers.

To define a critical number:

1. Open the **Critical Numbers Setup** form and execute Filter In Place.
2. To create a critical number from scratch, select **Actions > New**; or to create critical number from an existing one, copy and modify the existing critical number.
3. Specify this information:

   Critical Number
   Required. This must be a unique integer. If you do not specify an integer here, the system automatically assigns the next available integer.

   Active
   When selected (the default), this option activates the critical number and makes it functional. When cleared, this option deactivates the critical number.

   Snapshot
   Select this option to keep a snapshot history of the critical number.

   Description
   Optional. Specify a description of the critical number: what it represents, or how it is intended to be used. The value in this field is used as the caption on any gauges, etc. used to display the critical number's value.

Short Desc.
Optional. Specify a shorter description that the system can use in the Subject line of e-mails related to it.

Result Divisor
If the critical number is expected to return and display large values, and you want to divide those values before displaying them, specify the number by which to divide the value before it is displayed.

For example, if you expect the numbers to be retrieve to have values in the millions, such as 120,000,000, you can set this field to one million (1,000,000), and the resulting display shows that value as 120. In a case like this, you would want to set the **Description** field to **`(in millions)`** to clarify what value the number (120) actually represents.

4 On the**General** tab, specify the source, optional calculations, alert values to use, and goals to apply.

For more information, see the topic Making Critical Number General Tab Settings.

5 Optionally, use the **Drilldowns** tab to assign drilldowns to be used for this critical number.

Drilldowns are defined on the**Drilldowns Setup** form.

6 Optionally, use the **Categories** tab to assign the critical number to one or more categories.

For more information, see Critical Number Categories.

7 Optionally, use the **Users** tab to assign (or deny) permissions to individual users.

8 Optionally, use the **Groups** tab to assign (or deny) permissions to groups.

9 Optionally, use the **Static Parameters** tab to define name-value pairs that are used to allow end users to change critical number values without modifying the source code.

Note: Static parameter options are most effective when the source of data for the critical number is a stored procedure. When the data source is an IDO, the same end is accomplished much more easily using filters.

10 Optionally, use the **Input Parameters** tab to define and list filters that are used to determine what information the critical number is based on.

The **Sequence** number determines the order in which the IDO properties are evaluated during a critical number query. This data provides readable labels for the stored procedure parameters throughout the system.

11 Optionally, use the **Snapshots** tab to view a snapshot history of this critical number.

From this tab, you can launch the **Critical Number Snapshots** form to view details of the snapshots for the selected critical number.

You can also export the snapshot data to a CSV (comma-separated values) file that you can open in a spreadsheet application.

12 Save your changes.

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 Symbol
The default character is a capital **x**.

Warning Symbol
Specify a character to display on warning-related e-mails that are automatically generated. The default character is a minus symbol (**–**).

Goal Symbol
Specify a character to display on Goal-related e-mails that are automatically generated. The default character is a plus sign (**+**).

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. This setting does not prevent you from retrieving more records, it simply controls the batch size. So, if you want 12,500 records, and this option is set to 5,000, it requires three rounds of queries, instead of one, to retrieve the entire number of records (5,000 + 5,000 + 2,500).

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

Note: If the numbers are specific to a critical number, we recommend that you assign those to the individual critical number with which they are associated.

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

To set up multiple results for an IDO-based critical number:

- Build an IDO-based critical number.

  See Creating Critical Numbers on page 131.
- 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:

- Build the stored procedure-based critical number.

  See Creating Critical Numbers on page 131.

  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.
- 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 WBLoadCrAddSp to accomplish this:

```
CREATE PROCEDURE dbo.WBLoadCrAddSp (  @KPINum      WBKPINumType, @Cate
gory    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.

- 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.

- Choose the 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.
- 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.WBGetParm. 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:

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.WBGetParm(@CrNum, 'COStatusList'), 'POS')
SET @CoitemStatList = ISNULL(dbo.WBGetParm(@CrNum, 'COITEMStatusList'),
'PO')
SET @CredHold       = ISNULL(dbo.WBGetParm(@CrNum, 'CredHold'), 0)
SET @LateDays       = ISNULL(dbo.WBGetParm(@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.

Note:
- You can reach this form only by selecting the **Display Settings** option from the right-click menu of a critical number gauge display.
- Settings made on this form do not become part of the form definition. If you want to persist any changes beyond the current use/instance of the form, you must go into Design Mode and save them as part of the form definition.

To change the settings:

1. Right-click on any critical number gauge, and select **Display Settings**.
2. Use these fields to change any settings:

Field	Description / Comments
Critical Number	This field displays the ID number for the critical number value displayed in this gauge. You can, if you wish, select a different critical number for which to display data.

Field	Description / Comments
Date Range	Use this field to specify a date range for which to retrieve data: • **`All Dates`** • **`YTD`** (Year-to-date) • **`PTD`** (Period-to-date) **Note:** By default, a period is defined as one month. • **`WTD`** (Week-to-date)
Group	This field indicates what groups, if any, the critical number is assigned to. **Note:** If the critical number is configured to "Group By" some value, then the specific group can be specified in this field. So, for example, if you have the critical number set to display results by state, you might have one gauge set to display sales in Indiana and another gauge set to display sales in Ohio. You would then use this field to specify which gauge displays the results for which state.
As Of Date	Use this field to specify a starting date for the data being retrieved and displayed.
Drilldown	Use this field to specify the drilldown to be used with this critical number display.
Gauge Type	Use this field to specify what type of gauge display is to be presented to the user.
Goal Value	Use this field to change the upper or lower limit of the Goal range (depending on how the critical number is set up). **Note:** You cannot use operators in conjunction with the numbers in this field, only numbers or expressions that resolve to number values. For example, something like **`>=500`** would not be a valid value here. On the other hand, **`P(GoalValue)`** could be a valid value, as long as the value of the GoalValue property is a number.
Alert Value	Use this field to change the upper or lower limit of the Alert range (depending on how the critical number is set up). **Note:** You cannot use operators in conjunction with the numbers in this field, only numbers or expressions that resolve to number values. For example, something like **`>=500`** would not be a valid value here. On the other hand, **`P(GoalValue)`** could be a valid value, as long as the value of the GoalValue property is a number.

Field	Description / Comments
Caption Format	Specify the format to use for the gauge caption: • `Long:` The caption for the gauge displays the contents of the **Description** field on the **Critical Numbers Setup** form. • `Short`: The caption for the gauge displays the contents of the **Short Desc** field on the **Critical Numbers Setup** form. • `Default`: The type of caption that the gauge displays depends on the type of gauge being displayed. This option uses the gauge type to determine whether to use the **Description** or the **Short Desc** value. Some gauge types like Thermometer and Vertical LED are naturally narrow, so they use the **Short Desc** value by default. Other gauges, like the Horizontal Linear Gauge, are naturally wider, so they use the **Description** value by default. • `None`: No caption displays.
Sub-Caption Format	Specify which format to use for the gauge sub-caption: • `All`: This option displays the **Goal** setting, if used; the **Alert** setting, if used; and the **Actual** value of the critical number. Each value is labeled separately. • `Actual`: This option displays only the actual value of the critical number, with no label. • `None`: This option causes no sub-caption to be displayed at all.
Description	This read-only field displays the description for the critical number (usually like a title), as specified on the **General** tab of the **Critical Numbers Setup** form. This is also the same value used for a "Long" caption for the gauge display.
Calculation Definition	This read-only field displays the calculation definition as specified on the **General** tab of the **Critical Numbers Setup** form.

3 To save your changes and see the resulting display on the critical number gauge display, click **OK**.

About Critical Number Snapshots

Critical number snapshots are used to provide a history of critical numbers and show whether 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" was taken

- The actual (calculated) value of the critical number
- Whether the Alert option is selected and if so, a description of the alert range
- Whether the Goal option is selected and if so, a description of the goal value
- Whether a symbol is used to show how the actual value of the critical number compares to the goal and alert values set up for that number

Generating a Critical Number Snapshot

1. Open the **Snapshot Generation** form.
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.

Chapter 8: 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.

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

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.

A critical number drilldown is displayed in a DataView. As with other DataViews, you can manipulate the data and save layouts, whether you activate the drilldown from a critical number control or from the **Critical Numbers** form.

Setting Up a Critical Number Drilldown

The procedure to set up a critical number drilldown depends on the source for the critical number:

- For IDO-based drilldowns, see Setting Up an IDO Critical Number Drilldown on page 148.
- For stored procedure-based drilldowns, see Setting Up a Stored Procedure Critical Number Drilldown on page 142.

Setting Up a Stored Procedure Critical Number Drilldown

You can set up a drilldown for a critical number based on a stored procedure either by creating one new or by copying an existing one and then modifying it.

To set up a drilldown for a critical number based on a stored procedure:

1 Open the**Drilldowns Setup** form and execute Filter In Place.

2 In the **Drilldown** field, specify an unused integer to identify the drilldown.

  If you do not specify an integer here, the system automatically assigns the next available integer.

3 In the **Description** field, provide a descriptive name by which the drilldown and its use can be easily identified.

4 On the **General** tab, make the basic specifications for the drilldown.

  For more information, see Making General Tab Settings for a Drilldown.

5 Optionally, use the **Output Columns** tab to rearrange the order in which the output is to be displayed and the captions to be displayed for those columns.

  The data that is displayed on this tab is derived from the stored procedure identified in Step 4.

  For more information, see Setting Up Stored Procedure-based Drilldowns - Output Columns Tab.

6 Optionally, use the **Categories** tab to specify one or more categories to which the drilldown belongs.

  When you select a category, the **Category Description** field is populated automatically.

  For more information, see Critical Number Categories.

7 Optionally, use the **Sub Drilldowns** tab to specify one or more sub-drilldowns to use in conjunction with the selected drilldown.

  For more information, see Setting up a Sub-Drilldown Based on a Stored Procedure on page 150.

8 Optionally, use the **Static Parameters** tab to specify one or more name-value pairs

  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 this tab. You can then reference the static parameters when you set up the critical number/drilldown filter using the CRPARM() syntax.

  For example, a critical number based on an Account Balance stored procedure might have 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 users can change this value if they want to monitor a different account or if their cash account uses a different account number.

9 Optionally, use the **Input Parameters** tab to provide values that can be used in processing sub-drilldowns, especially.

10 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.

11 Create your stored procedure in your preferred code 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
```

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.

This example shows Customer Order Past Due:

```
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 = NUL
LIF(@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 an IDO Critical Number Drilldown

You can set up a drilldown for an IDO-based critical number either by creating one new or by copying an existing one and then modifying it.

1. Open the **Drilldowns Setup** form and execute Filter In Place.
2. In the **Drilldown** field, specify an unused integer to identify the drilldown. If you do not specify an integer here, the system automatically assigns the next available integer.
3. In the **Description** field, provide a descriptive name by which the drilldown and its use can be easily identified.
4. On the **General** tab, make the basic specifications for the drilldown.

   For more information, see Making General Tab Settings for a Drilldown.
5. Optionally, set up the IDO source for the drilldown.

   For more information, see Setting Up the IDO Source for a Drilldown.
6. Optionally, use the **Output Columns** tab to rearrange the order in which the output is to be displayed and the captions to be displayed for those columns. The data that is displayed on this tab is derived from the IDO source set up in Step 5.

   For more information, see Setting Up IDO-Based Critical Number Drilldowns - Output Columns Tab.
7. Optionally, use the **Categories** tab to specify one or more categories to which the drilldown belongs. When you select a category, the **Category Description** field is populated automatically.

   For more information, see Critical Number Categories.
8. Optionally, use the **Sub Drilldowns** tab to specify one or more sub-drilldowns to use in conjunction with the selected drilldown.

   For more information, see Setting Up an IDO-Based Sub Drilldown on page 150.

9 Optionally, use the **Static Parameters** tab to specify one or more name-value pairs

   Note: Static parameter options are most effective when the source of data for the critical number is a stored procedure. When the data source is an IDO, the same end is accomplished much more easily using filters.

10 Optionally, use the **Input Parameters** tab to rearrange the order in which input parameters are queried for data and provide descriptions of each.

   The data that is displayed on this tab is derived from the IDO source set up in Step 5.

11 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 your changes.

Setting Up a Critical Number/Drilldown IDO Filter

1 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 for regular orders.

   Note: You cannot open this form directly.

2 In the **Property Name** field, specify the IDO property for which you want to set up the filter. This field is not available if the comparison type is `CN Parameter Has Value`.

3 Specify the operator to use for the filter. If you select `IS NULL` or `IS NOT NULL`, the comparison field and value need not be specified. This field is not available if the comparison type is **CN Parameter Has Value**.

4 Select the comparison type:
   - `Literal`: The property value is compared to the literal value you specify in the last field on this row.
   - `Property Name`: The property value is compared to another property in the same IDO. The other property is specified in the last field on this row.
   - `Critical Number Parameter`: The property value is compared to the value of a static parameter defined on the **Critical Numbers** form. The name of that parameter is then specified in the last field on this row.
   - `CN Parameter Has Value`: The property value is compared to a Critical Number parameter value. That value is specified in the last field on this row. If this option is selected, then the **Property Name** and operator fields are not available.

5 In the last field on this row, specify the comparison value to be used for the corresponding comparison type.

6 Optionally, if you are using multiple filter clauses, to instruct the system to treat the clauses as Boolean OR comparisons rather than Boolean AND comparisons, select **OR Instead of AND with Previous Clause**.

7 To add the filter clause to the list of clauses in the display panel, click **Add**. The clause is added to the list, formatted with the proper syntax.

8 Optionally, use Steps 2 through 7 to specify additional filter clauses.

9 Optionally, to remove all filter clauses and start over, click **Remove**.

   Note: You cannot selectively remove individual filter clauses: If you choose to remove one, you remove them all.

10 Click **OK**.

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.

Setting Up an IDO-Based Sub Drilldown

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

To associate a new drilldown as a sub drilldown:

1 Create the drilldown.

2 On the **Drilldowns Setup** form, 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-Drilldown Based on a Stored Procedure

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

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.

To associate a new drilldown as a sub drilldown:

1 Create the drilldown.

   See Setting Up a Stored Procedure-based Critical Number Drilldown on page 142.

2 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.

3. 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.
4. Launch the **Drilldowns Setup** form.
5. Navigate to the top level drilldown.
6. On the **Sub Drilldowns** tab, specify the new drilldown and a description for it.
7. Save the record.

Example:

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
```

Chapter 9: DataViews

About DataViews

DataViews are advanced data grids that you can create to query a custom set of data, quickly and easily. The basic DataView presentation is similar to an Excel spreadsheet, displaying data in columns.

Among other things, you can perform these actions:

- Use DataViews to create special groupings of data and summaries, optionally with post-query filters.
- Merge data from two levels of data and present them as one record.
- Use the data returned from various sources to create your own custom columns that include calculations and calculated summaries.
- Navigate to the related data maintenance form by clicking **Details** on the right-click context menu.

DataView Sources and Types

The data for a DataView can come from a variety of sources, each resulting in one of four types of DataView (or DataView-like) displays:

- Form DataViews can be created from any form that displays data.
    - These DataViews are launched by clicking the **DataView** button in the toolbar.
    - These DataViews use the data from the form as the source.
    - It does not matter whether the data was filtered before displaying on the form or loaded using a custom load method. But keep in mind that the user can see in the DataView only the data that they can access on the form.
    - You can manipulate the layout for a DataView to view the form data in ways not possible on the form.
    - Layouts created from these DataViews can be saved and applied to the data from the same form again in the future.
    - There is a toolbar icon that you can use to send the data in the current collection to DataView. If this icon is hidden in your toolbar, you can use the Theme editor to display it.
- Predefined DataViews are constructed using the DataViews setup process and are available only to users who have access.
    - These DataViews are typically launched from the **DataViews** form.
    - They require more setup than form DataViews (using the **DataViews Setup** form).
    - Predefined DataViews provide greater control over which IDOs are accessed and which properties are shown than do Form DataViews.
    - Permissions made on the **DataViews Setup** form control which DataViews a given user can see.
    - Predefined DataViews can be embedded in forms. This is done by implementing a DataView component from the Toolbox in Design Mode.

    - Predefined DataViews can be used to produce custom reports. These reports can be treated as system reports, with Vendor-level protections and associated report criteria forms.

- Critical Number Drilldowns are specialized DataViews that provide details about what makes up a Critical Number.
    - Drilldowns are launched by double-clicking a Critical Numbers form or gauge.
    - The drilldown controls specifically which properties are displayed.
    - These drilldowns inherit their permissions from Critical Numbers licensing and provisioning.
    - They have the same layout capabilities as other DataViews.
    - Drilldowns are configured using the **Drilldowns Setup** form.
    - Both IDO collection and stored procedure sources are supported.
    - When used on Critical Numbers forms, row coloring is supported for goals and alerts.

- DataSearches are specialized searches in which you can search across predefined data sources for specific values.
    - DataSearches are launched either by clicking the DataSearch toolbar button or by opening the **DataSearch** form.
    - The search results are displayed in a DataView-like display.
    - Compared to the typical DataView display, however, the DataSearch display, while similar, is much more limited. Basically, you can only expand the data records, and you can reorder the columns.
    - Sources of data to be used for DataSearches are defined using the **DataSearch Source Setup** form. This form provides control over which properties are searched and which users or groups can see what data sources.

DataView Results

When data is returned from a DataView query, the results are typically displayed in a basic grid on the appropriate DataView results form. There, depending on what kind of DataView query was performed, you can reorganize the data, decide what data displays, and take other actions with the results, such as printing a quick, on-the-fly report, or export the data to a spreadsheet.

See About DataView Results.

DataView Layouts

To organize and present the data that is included in a DataView, you can create layouts. Layouts can eliminate the redundancy of grouping and sorting the results every time you want to view the data.

These layouts are created by employing a number of options. You can perform these actions:

- Rearrange, reorder, and resize the columns
- Define data groupings
- Create custom columns, which can include "on the fly" calculations
- Display or hide selected columns
- Merge different levels of data
- Define summaries

Once you have the DataView layout organized and the data presented the way you want it, you can save these layouts for future use and reference. You can even use these layouts to create your own custom reports.

As with form customization, DataViews support user-level, group-level, and site-level layouts. Access to these layouts can also be controlled using filters and user permission settings.

See About DataView Layouts on page 163.

Setting up a new DataView

To set up (create) a new predefined DataView:

1. On the **DataViews Setup** form, execute Filter In Place.
2. Click in the Grid View, and then click in the auto-insert (last, empty) row.
3. In the **DataView** field, specify a name that best describes the data to be presented.
4. Optionally, to prevent modifications to the DataView by users other than those with Vendor Developer permissions, select the **System Record** option.

   This option allows you to protect the structure of DataViews that you deliver to customers/users. It prevents others from deleting or modifying any content you provide, while also allowing customers to add their own content.
5. Optionally, to designate a report caption/title, in the **Caption Override** field, specify the caption.

   This can be a translatable string or a literal value.

   When specified, this string replaces the default system-generated caption on DataView report outputs.

   Note: You can also specify a layout-specific caption override, by setting the **Caption Override** field on the **Layouts** tab. If both that and this field have caption overrides, the layout-specific override takes precedence.
6. Optionally, use the **Criteria Form** drop-down list to select a form to use to collect report criteria.

   This option is used primarily when the report requires a significant number of option specifications for data to be collected. When that is a low number, you typically prompt for each option individually; but when there is a significant number, it is easier and faster for the user to specify the data to be collected using a report criteria form.
7. On the **General** tab, specify the required and optional information for each IDO that is to be included in the DataView queries.

   See Specifying DataView setup information - General tab on page 165.
8. Optionally, after specifying each IDO, to make additional IDO specifications for the DataView, click **IDO Setup** and follow the procedure in the topic Setting additional IDO specifications for a DataView on page 167.
9. Optionally, use the options on the **Input Parameters** tab to add input values or ranges.

   See Specifying DataView setup information - Input Parameters tab on page 169.
10. Optionally, use the options on the **User Permissions** tab to control who has access to this DataView.

    See Specifying DataViews setup information - User Permissions tab on page 170.
11. Optionally, view information about or make copies of layouts on the **Layouts** tab.

    See Specifying DataView setup information - Layouts tab on page 171.

12 Optionally, use the **Report Options** tab fields to set the report orientation and to select what regions are to be displayed on the report.

See Specifying DataView setup information - Report Options tab on page 172.

13 Save your work.

Note: If the intended use for this DataView is to be used for report output, you should create an associated report criteria form.

Displaying a Predefined DataView

Use the **DataViews** form to launch the **DataView Results** form and view predefined DataViews and layouts. Predefined DataViews are DataViews that have been set up for specific uses in the system. You can use the results on these DataViews to drive ad hoc reporting.

To display a predefined DataView:

1 Open the **DataViews** form.
2 To display the list of predefined DataViews, execute Filter In Place.
3 Select a DataView from the list.
4 If multiple layouts exist for the DataView, select a layout.

To help determine which layout to view, examine the column values for each layout:

Column	Description
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: Vendor, Site, Group, or User.
Scope Name	This field specifies the group or user for which this layout is designed. Otherwise, this field displays **`[NULL]`**.
Default	When more than one layout exists for the same scope type and name, this field is selected for the layout that takes precedence .

5 To open the **DataView Results** form for the selected DataView layout, click **Launch**.

Creating Custom Columns for DataViews

Use custom columns to merge and display data from various sources and to create columns on which you want to perform operations and calculations on data in ways not possible on the original forms.

Use the **Custom Columns**dialog box to create custom columns for a DataView:

1 While DataView results are displayed, select **Display > Custom Columns**.
2 Verify that the **Collection** field shows the IDO collection from which the data is being retrieved.
3 In the **Name** field, specify the name to assign to the custom column.

   Note: This name is the label that will be used for the custom column header. Take care when specifying this name, because you cannot change it later. If you decide to change the name/label later, you must delete the existing column and create it again with the new name.

4 In the **Data Type** field, specify the type of data to use for the custom column display.
5 Click **Add**.
6 Use the **Advanced Editor** to define the formula or expression that is to specify what is displayed in the column.
7 After you finish in the **Advanced Editor**, click **OK**.

The DataView grid immediately shows the new column with the results of the formula or expression.

Specifying Summaries for DataViews

Use specified summaries to display aggregate information about the data represented in a DataView. This aggregate information can include either standard summaries, such as Average, Count, Maximum, Minimum, and Sum; or custom summaries that you define.

1 When DataView results are shown, select **Display** > **Show Summaries**.
2 Select a column on which you want to base a summary and click the "sigma" button.
3 In the **Select Summaries** dialog box, specify one or more standard summaries, one or more custom summaries, or both.

   To specify a standard summary, depending on the data type of the column you used to launch the **Select Summaries** dialog box, you can select any combination, all, or none of these options:

   - Average: Calculates the mean average of the values retrieved for that column. This option is enabled only for numeric data types.
   - Count: Shows the total number of all the records retrieved.
   - Maximum: Shows the maximum value represented in that column. If the value is a string or text value, standard alphanumeric valuation is used.
   - Minimum: Shows the minimum value represented in that column. If the value is a string or text value, standard alphanumeric valuation is used.
   - Sum: Adds the values of all records retrieved for that column. This option is enabled only for numeric data types.

   To specify a new custom summary in the **Select Summaries** dialog box:

   a In the **Name** field, specify the name to assign to the custom summary.

      Note: This is the label that will be used for the custom summary caption. Take care when specifying this name, because you cannot change it later. If you decide to change the name/caption later, you must delete the existing summary specification and re-create it with the new name.

   b Click **Add**.

c Use the **Advanced Editor** to define the formula or expression that is to specify what is displayed in the column.

After you finish in the **Advanced Editor**, click **OK**.

The summary bar at the bottom of the DataView grid immediately displays the specified summaries.

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.

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

Specify this information on the **DataView Results** form:

Form Menu	Description
Layout	You can save changes to the current layout or save it as a new layout with a different name. In either case, you can designate a layout as the default layout for a DataView. You can also delete and select existing layouts.

Form Menu	Description
Display	Use these options to manipulate the data in the results grid: • **Expand All**: This option displays all rows of data. • **Collapse All**: This option displays only the top level rows of data. • **Choose Columns**: This option launches the **Choose Columns** dialog box, in which you can designate which columns are to display and which are to be hidden. • **Custom Columns**: This option launches the Custom Columns dialog box, in which you can create new columns from existing columns in the data results. • **Show/Hide Options**: These options display or hide the **Group By Area** in the DataView header, as well as the Summary, Filter, and Pin Column icons in the column header. • **Show/Hide Group By**: This option displays or hides the **Group By Area** in the DataView header. • **Show/Hide Column Scroll Lock**: This option displays or hides the Pin Column icons in the column headers. • **Show/Hide Filters**: This option displays or hides the Filter icons in the column headers. • **Show/Hide Summaries**: This option displays or hides the Summary icons in the column headers. • **Show/Hide Extended Captions**: This option displays or hides extended captions in the column headers. An extended caption consists of any default caption (or caption override), followed by the property name in parentheses.
Print	This menu consists of these entries: • **Print Preview** - This option generates a preview of the DataView as it will appear when printed. This preview displays in a separate window. • **Print** - This option launches the Print dialog box, which allows you to print the DataView display to any local system-enabled printer.
Send To	Use this menu to export the DataView output. This option uses the current state of the DataView for export. Features such as column order and visibility, filters, and summaries are preserved and supported in the output. Output options include these: • **Excel** - This option formats the output as a spreadsheet file and then opens it in Excel (or other compatible spreadsheet application). • **XPS** - This option converts the output to XPS format and saves it as an XPS file. • **PDF** - This option converts the output to Portable Document Format (PDF) and saves it as a PDF file. **Note:** The Group feature is not supported in the web client. If a DataView uses grouping, you cannot export the data from a web client. Also, the web client supports export only to Excel files, not XPS.

Form Menu	Description
Data	Use the options on this menu to process the data displayed by the DataView. The menu has these options: • **Refresh** - This option requeries the data using the same search criteria as the current display. • **Get More Rows** - When the number of records available exceeds the record cap for a query, this option allows you to requery and return more records. • **Prompt for Inputs** - For DataViews that have input parameters with no supplied values, this option allows you to perform a new query using different input parameter values.
Setup	This option launches the **DataViews Setup** form pre-filtered for the current DataView.

Note: If you do not have permission to see the results, an authorization message is displayed instead of the results.

About DataView Layouts

Once a basic DataView has been created, users who have access to them can rearrange and further organize the DataView display. You can perform these additional actions:

- Columns can be moved or hidden.
- Summaries of data can be created.
- Data can be grouped and merged.
- Retrieved data can be further filtered.
- Custom columns can be created and treated like any other columns. These custom columns can include calculations that modify and manipulate the data so as to present additional values not available from just the original data source.

Use these actions to rearrange, reorganize, and manipulate the data that was retrieved.

After the information is organized and displayed as desired, the resulting presentation is considered a layout.

Generally, a layout is only available as long as the DataView is still open and not modified further. However, any particular layout can be saved and reused.

Furthermore, any given DataView can have virtually any number of layouts created from it, simply by using the techniques and practices already mentioned. This means that a single DataView can have multiple layouts and many uses.

Note: DataSearch layouts are the exception to this rule. You can save DataSearch layouts, but you can only have one layout per DataSearch Source.

Layouts created and saved by Vendor Developers have a scope of "Vendor" associated with them. Users who have Site Developer editing permissions can save layouts at the Site, Group, and/or User scope level. Individual users with DataViews access can normally only save layouts at their own user level.

Layouts can be copied and used as the basis for yet other layouts. For example, the copy of a layout might have additional custom columns added to created calculated values. This version of the layout might then have a scope of `User` assigned and be assigned to a single manager.

Copying a DataView Layout

You can copy a DataView layout either from the **DataView Layouts** form or from the **Layouts** tab of the **DataViews Setup** form.

To copy a DataView layout:

1 From the list of saved layouts, select the one you want to copy.
2 Click **Copy Layout**.
3 Optionally, use the **Layout** field to rename the copied layout.
4 Optionally, change the **Scope Type** to the desired level.
5 If you changed the **Scope Type** to either `Group` or `User`, use the appropriate drop-down list to specify the desired group or user.
6 Optionally, if there are multiple layouts with the same name and scope type, select the Default check box for whichever layout should be considered as the default layout.
7 Save your work.

Setting Up Predefined DataViews

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

The definition for a predefined DataView consists of up to four basic sets of options:

- The general settings are the fundamental settings that define the DataView. These settings are made on the **General** tab:
  - The DataView name
  - The IDOs that the DataView is set to query
  - How many records the query is limited to
  - The definitions of any filters to be used in the query
  - Other related information

  For more information, see Setting up a new DataView on page 158.

- Input parameters are considered optional and are set using the **Input Parameters** tab. Input parameters are used as filtering mechanisms to limit the amount and type of data retrieved. These parameters can be especially useful in setting up DataViews as sources for formal reports.
- You can control who has access to what DataViews by setting user and group permissions on the **User Permissions** tab.
- You can view information about and make copies of layouts, using the **Layouts** tab.
- Finally, you can determine the way the report will print, using the options on the **Report Options** tab.
  Note: A predefined DataView is first processed as a temporary report the same as a Report type form. Because of this, you can use these options to specify what options are to appear on the final report output.

Specifying DataView setup information - General tab

When you set up a predefined DataView, specify this information on the **General** tab of the **DataViews Setup** form.

Note: You can specify multiple data sources to be queried in the DataView. Specify individual information for each data source.

Option	Required or Optional?	Description/Comments
IDO Setup	Optional	Use this button to launch the **DataView IDO Setup** form and specify exactly what data you want to use from a specified IDO. You can use this button/form for each IDO that you specify for a DataView, to define and limit exactly what data from each IDO is to be queried. See Setting additional IDO specifications for a DataView on page 167.
Source Type	Required	Specify whether the data source is to be an **IDO Collection** or a custom load **IDO Method**.
IDO	Required	Specify what IDO is to be the source for the data to be returned.
IDO Alias	Required	Verify that the IDO alias is what you want. This value is generated automatically. You can modify the recommended alias.
Source Name	Required when enabled	Specify the name of the custom load IDO method being used as the data source. This field is enabled only if **IDO Method** is selected as the **Source Type**.

Option	Required or Optional?	Description/Comments
Parent IDO	See Description/Comments.	Use this field to specify an IDO collection that is to be considered the "parent" of the subcollection specified in this row. This field is enabled only if **IDO Collection** is selected as the **Source Type**. It is required only if the specified IDO collection is a subcollection.
Record Cap	Optional	Use this field to specify the maximum number of records to return when the DataView query is performed. Select from these options: • **Use System Setting** - The system record cap setting determines the maximum number of records that can be returned. • **Use Specified Max** - This option allows you to set your own maximum for the number of records to be returned. When selected, this option makes visible and enables the **Record Cap Value** field. • **Retrieve All** - This option instructs the system to retrieve all available records, regardless of any record cap settings elsewhere on the system.
Record Cap Value	See Description/Comments.	This column is visible and enabled only when the **Record Cap** setting is **Use Specified Max**. By default, this field contains the system setting for the record cap. You can change it to whatever value you choose.
Filter	Optional	Specify a filter for the system to use when performing the query. This option is enabled only when **Source Type** is **IDO Collection**. To filter data when **Source Type** is **IDO Method**, you must specify the filter parameters within the method itself.
Order By	Optional	Optionally, specify which property is to be used to sort the retrieved results. This option is enabled only when **Source Type** is **IDO Collection**.

Option	Required or Optional?	Description/Comments
Link Type	Optional	Specify whether the parent-child link is to be a multi-level or single-level link. **Note:** This option is enabled only if a **Parent IDO** is specified. • The **Multi Level** option (default) nests and indents the child IDO information under the parent information. • The **Single Level** option links and presents the parent and child IDO information at the same level.
Link By	Optional	Specify how the subcollection is linked to the parent IDO collection. **Note:** This option is enabled only if a **Parent IDO** is specified. The syntax is the same as described in IDO Link By Editor.
System Record	Optional	To prevent modifications to the setup for the selected IDO by users other than those with Vendor Developer permissions, select this option. This option is slightly different from the System Record option for the DataView itself, in that it protects only the selected IDO setup. The exceptions to this rule are the Record Cap settings and the Show Notes settings.
Show External Notes	Optional	To display any external notes that might be attached to the DataView, select this option. When this option is selected, external notes are displayed as child layers in the DataView, underneath the IDO.
Show Internal Notes	Optional	To display any internal notes that might be attached to the DataView, select this option. When this option is selected, internal notes are displayed as child layers in the DataView, underneath the IDO.

Setting additional IDO specifications for a DataView

When you create a predefined DataView, you can set up the associated IDO so that only the desired data is retrieved when the DataView is displayed. This ensures that you minimize the retrieval time and get only the data that you really need.

Note: This information applies only to predefined DataViews. The data for other types of DataView displays is controlled by the source from which the display was launched.

Also, the information in this topic is closely related to the information in the topic Specifying DataView setup information - General tab on page 165. In general, when information is presented in that topic, it is not repeated here. Only additional information and the expanded capabilities of the **DataView IDO Setup** form is presented here.

To configure additional IDO specifications for a DataView, use these guidelines:

- **IDO** section - Depending on the **Source Type**, you can change or modify several of the settings in this section. You can also set or modify these settings on the General tab of the **DataViews Setup** form.
- **Link By** section - As with the **Link By** option on the **DataViews Setup** form, these options are enabled only when a **Parent IDO** is specified on that form.

  This form, however, provides additional specification options. You can specify which properties are to be used to link and display data. Do this by specifying a **Parent Property** and a **Child Property** and then clicking **Add**.
- **Order By** section - As with the **Order By** option on the **DataViews Setup** form, these options are enabled only when the **Source Type** is **IDO Collection**, as specified on that form.

  This form, however, provides additional specification options. You can specify properties by which to sort the data that is displayed. Do this by specifying one or more properties by which to sort the data and whether the data is to be displayed in **Ascending** or **Descending** order; and then clicking **Add**.

  Note: If a record cap is in use, the **Order By** setting determines which records are selected for retrieval first.
- **Properties** section - You can specify exactly which properties are to be included in the data query by selecting them in this section. Properties that are not "**Selected**" are not included in the return results for the data query.

  In addition to selecting which properties to include in the data query, you can also view and/or specify these settings for each property:

Setting	Description
Description	Read-only. This field displays any description provided for the property, as specified on the **IDO Properties** form.
Default Caption	Read-only. This field displays any default caption currently defined for the property, as specified on the **IDO Properties** form. You can override the default caption, using the **Caption Override** field.
Caption Override	Use this field to override the default caption for a property. Whatever you provide as a caption override is what displays as the column header for that property in the **DataView Results** form.
System Record	When selected, this option prevents anyone other than a Vendor Developer from making modifications to or deleting a property specification for the DataView.

Specifying DataView setup information - Input Parameters tab

When you set up a predefined DataView, you can specify this information on the **Input Parameters** tab of the **DataViews Setup** form.
Note: The options on this tab can be especially helpful when setting up a DataView to be used to create reports. You can use input parameters to specify exactly what data can be used for query ranges or other return results, using a report criteria form associated with the DataView.

Column	Description
Sequence	These integers indicate the order in which the specified input parameters are queried. You can assign this number at the time of the parameter's creation. Once the parameter has been saved, you cannot change it.
Property Name	Use the drop-down list to specify the IDO property to be used for the input parameter. When you select a property in this field, the **Description** field is automatically populated with the default value for the property, if one exists. You can modify the value in the **Description** field, if desired. **Note:** If the **Source Type** on the **General** tab is set to **IDO Method**, this field is disabled.
Operator	When used, this field specifies an operation to use in setting a range for the data to be retrieved. When two such operators are used in tandem, you can set an upper range limit and a lower range limit. For example, suppose you want to retrieve the values of all customers whose names begin with the letter 'C'. You could create one parameter using the operator **Greater Than Or Equal To** with a **Description** value of `C`. You could then create a second parameter using the operator **Less Than**, with a **Description** value of `D`. By applying both of these parameters to the **Property Name** of `CustName`, you effectively set a range for customers whose names begin with the letter 'C'. See the help topic DataView Input Parameter Operators. **Note:** If the **Source Type** on the **General** tab is set to **IDO Method**, this field is disabled.

Column	Description
Description	Specify a string that describes this parameter. This string is used when prompting for this parameter, so that the user has an idea of what the parameter is. In sense, you can consider this a prompt string or label string. This string can be a literal string (for example, `Starting Customer`) or it can be the name of a translatable string (for example, `sStartingCustomer`). When translatable strings are used, they are translated at run time. If the DataView is self-prompting, then this string is used as the label for the input parameter on the **DataView Inputs** form. If the DataView component is embedded in a form, then this string is used to label the input parameter field within the DataView component binding.
End of Day	This option is enabled only when the specified property is a date/time property. When selected, this option causes the time on a Date/Time property value to be set to the "end of day," which is defined in the system as `11:59:59.99 PM`. When cleared, the system uses the system date/time that the user performed the query. The use of this option allows you to set very precise date/time ranges. **Note:** If the **Source Type** on the **General** tab is set to **IDO Method**, this field is disabled.

Specifying DataViews setup information - User Permissions tab

Use the **User Permissions** tab on the **DataViews Setup** form to specify who can access the DataView that is being created or modified. You can specify permissions at the user level, the group level, or a combination of the two.

Column	Description
User Name	Specify users who are to have access to the DataView. The list is populated from the list of users in the system.
[User] **Description**	(Read-only) This field displays the description associated with the specified user, as specified on the **Users** form.
Group Name	Specify user groups that are to have access to the DataView. The list is populated from the list of user groups in the system.
[Group] **Description**	(Read-only) This field displays the description associated with the specified user group, as specified on the **Groups** form.

Specifying DataView setup information - Layouts tab

When you set up a predefined DataView, you can specify this information on the **Layouts** tab of the **DataViews Setup** form.

Note: You can also use this tab to create copies of layouts. These duplicate layouts can each have a different scope and can be designated as the default version of the layout if desired.

Field/Option	Description
Copy Layout	To create a copy of a layout that you can use as the basis of a new layout, click this button. When you copy a layout, the copy initially has all the same field values, except for the **Scope Type**, which defaults to **Site**.
Layout	Use this field to name and identify a new layout. For layouts that have already been saved, this field is read-only and shows the name of the layout as it was saved.
Scope Type	Use this field to specify the level of scope to which the layout is available. The default scope for a copied layout is **Site**. If you specify a scope of either **Group** or **User**, you must also specify a **Group Name** or **User Name**, respectively. If you have Vendor Developer editing permissions, and you want to define the scope type as **Vendor Default**, enter `0` (zero) in this field.
Scope Name	If you specify either **Group** or **User** in the **Scope Type** field, you must specify the name of the applicable user group or the individual user in this field. **Note:** If **Group** is specified as the **Scope Type**, this column's header displays as **Group Name**. If **User** is specified as the **Scope Type**, this column's header displays as **User Name**.
Default	When there are multiple layouts with the same **Scope Type** and **Scope Name**, this option specifies the default layout. The **Default** layout option is set using the **DataView Layouts** form. On this form, the option is read-only.
Caption Override	Optionally, to designate a caption/title to use for the selected layout, specify the caption. This can be a translatable string or a literal value. When specified, this string replaces the default system-generated caption on DataView report outputs. **Note:** You can specify a DataView-level caption override, by setting the **Caption Override** field at the top of the form. If both fields have caption overrides, the layout-specific override here takes precedence.
Report Orientation	Use this field to set the orientation (**Landscape** or **Portrait**) for the selected layout. **Note:** You can specify a DataView-level orientation, by setting the **Report Orientation** field at the top of the form. If both fields have orientations specified, the layout-specific setting here takes precedence.

Specifying DataView setup information - Report Options tab

When you set up a predefined DataView, you can specify this information on the **Report Options** tab of the **DataViews Setup** form.

Fields/Options	Description
Report Orientation	Sets the report output to either a Portrait or Landscape orientation.
• **Display Report Header** • **Display Page Header and Footer** • **Repeat Headers on New Page** • **Repeat Headers on Collection Change** • **Insert Page Break Between Groups** • **Reset Page Number Between Groups** • **Can Grow**	Because the predefined report output is first determined by a temporary Report form layout, these settings are the same as those used for the Report form. **Note:** This option applies only to reports running as background tasks (TaskMan, runreport.exe). It does not apply to DataViews running in the foreground.

Copying a DataView to Create a New One

You can copy an existing DataView that is close to what you want, and then modify it and save it as a new DataView.

To create a new DataView from an existing one:

1. Optionally, with the **DataViews Setup** form open, select the DataView that you want to copy, and click **Copy DataView**.
2. In the **DataViews Copy** form, verify that the **Current DataView** is the one that you want to copy.
3. In the **New DataView** field, specify a name for the new DataView.
4. Click **Copy DataView**.
5. When the confirmation message displays, click **OK**.

   The **DataViews Copy** form closes, and the new DataView copy is added to the list of DataViews in the **DataViews Setup** form.
6. Select your new DataView and modify it as desired.

   Use the procedure in Setting up a new DataView on page 158, starting with the step to specify the required and optional information for each IDO.

Setting Up a DataView Filter

1. To launch the **DataView Filter Setup** form, click **Filter** on the **DataView IDO Setup** form.

   Note: You cannot open this form directly.

2. In the **Property Name** field, specify the IDO property for which you want to set up the filter.
3. Specify the operator to use for the filter. If you select **`IS NULL`** or **`IS NOT NULL`**, the comparison field and value need not be specified.
4. Select the comparison type:
   - **`Literal`**: The property value is compared to a hard-coded value.
   - **`DataView Property`**: The property value is compared to another property value from the DataView.

     This can be useful when you want to use a comparison of two properties to control what records are returned. For example, if you wanted to see which orders do not have the full number of items shipped, you could set up the property comparison **`QtyShipped < QtyOrdered`**. Or suppose you wanted to know which orders were not shipped on time, you could set up the property comparison **`ShipDate > DueDate`**.
   - **`Form Property`**: Use this option in cases where the DataView is embedded on a form and you want to filter the DataView based on properties from the form's IDO collections.

     For example, you could use this option to filter the DataView's customer number property by the form's customer number property. When you then add the filter clause, the resulting syntax might look something like this: **`CustNum = FP(CustNum)`**

5. In the last field on this row, specify the comparison value to be used for the corresponding comparison type.
6. Optionally, use Steps 2 through 5 to specify additional filter clauses.
7. Optionally, if using multiple filter clauses, to instruct the system to treat the clauses as Boolean OR comparisons rather than Boolean AND comparisons, select the check box labeled **`OR Instead of AND with Previous Clause`**.
8. To add the filter clause to the list of clauses in the display panel, click **Add**.

   The clause is added to the list, formatted with the proper syntax.

9. Optionally, to remove all filter clauses and start over, click **Remove**.

   Note: You cannot selectively remove individual filter clauses: If you choose to remove one, you remove them all.

10. Click **OK**.

For complex comparison logic, it might be necessary to add parentheses around filter clauses, to make sure they are evaluated properly. To accomplish this task, manually edit the data in the display list.

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 Setup** form to set up right-click menu actions for components. You can specify the forms to open, executables to run, etc. 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 can be available on any component that display a customer number field. You can also associate a menu action with a specific IDO or property to limit the action's availability to a more specific set of components.

Note: The options you specify in the dynamic menu list are displayed alphabetically within Action Type and are displayed below the static menu options.

Each time a user right-clicks on a component, the system dynamically builds the menu options based on the menu actions that are defined for the class, IDO, and property of the component value. The user can perform any of these types of actions that are defined:

- Launch a specific form that is filtered to show values from the selected component value.
- Run an executable program and pass it parameters that are values from the selected component value.
- Launch another component value.
- Perform a global search, which are available on all columns, that launches the **DataSearch** form that is filtered with values from the selected component value.

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 forms and components, use the Applies To section of the **DataView Actions Setup** 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 component that is associated with a customer number. However, if you specify the **IDO** as **`WBFSCustomers`**, then only those components that are built using the WBFSCustomers IDO display the menu action.

In this section, you can specify this information:

- Specify the**Scope** level to which this action applies: Vendor, Site, Group, or User. If the scope is Group or User, specify the group name or user name.
- 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:

- 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`**, you must 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`**, you must specify the name of the form to open and the **Initial Command**: Refresh, Add, or 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 list of variables must be separated by a comma and must contain 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 component, 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 form 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. See "Substitution Keywords" below.
  - If the **Action Type** is **`Run Executable`**, click **Browse** and select the path and file name of the executable program that you want to run when the action is selected.
    Note: The executable program runs on the client. If users that select this action cannot access the executable file on their local computers using the path you specified here, an error message is displayed.
- 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 is 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 component 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.

See Setting Up a DataView Actions Filter on page 176.

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 as they work in design mode, except that if the component is a DataView. 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 form
- To pass a value from the form to an executable in a command line parameter

Setting Up a DataView Actions Filter

To build a filter to use on a DataView action:

1. In the **DataView Actions Filter Setup** form, select the property of the IDO to be evaluated.

   For a Run DataView action type, this is the primary collection of the predefined DataView. For a Run Form action type, this is the primary collection on the target form.

   If this form is launched from the **DataView Actions Setup** form with a Run Form action type selected, the **Property Name** drop-down list shows all bound and derived properties in the IDO, not just those that are bound to the form.

2. Select the operation method.

   If you select **`Is Null`** or **`Is Not Null`**, do not specify anything in the **Comparison** field and value.

3. Select the comparison type:
   - **`Literal`**: The property is compared to a specified value.
   - **`Source DataView Property`**: Use this IDO property from the source DataView (where the user right-clicked) to filter the form or DataView that is opened by the action. For example, if the user right-clicks on a customer number and selects an **Action** menu option that opens the **Customer Orders** form, the additional filter created here could include the Ship To (CustSeq) property as part of the filter, like this: **`CustSeq=FP(CustSeq)`**.
   - **`Target DataView Property`**: Use this IDO property from the target DataView (the DataView that will be opened by the action) to filter the DataView that is opened. For example, if the action opens a **Customer Order Lines** DataView, this filter could show only unfilled order lines, where QtyOrdered > QtyShipped.
   - **`Target Form Property`**: Use this IDO property from the target form (the form that will be opened by the action) to filter the form that is opened. For example, you could add QtyShipped < QtyOrdered to the filter when the action opens the **Customer Order Lines** form.

4. If you selected **`Literal`**, specify a comparison value.
5. Click**Add** to translate the information from the fields into filter syntax and show the filter expression in the editor box.
6. Click **OK** to save changes.

Notes:

- Click **Remove** to clear the content in the editor box.
- For complex comparison logic, it may be necessary to manually edit the text in the editor box to add parentheses around or and and statements, to ensure that the filter is evaluated properly.
- To better understand how filters can be used, look at the vendor level DataViews that are provided as part of the application.

Chapter 10: DataSearch

About DataSearch

The DataSearch feature allows you to set up and perform data searches that can span across the entire system. You can, for example, set up DataSearch to search across all your application system files and return any records that have the string "young" in them. It does not matter whether those records are found among the customer records, the order and order line records, purchase order records, or part and version number records.

DataSearch Sources

Before a DataSearch can be performed, you must have at least one DataSearch Source set up. The DataSearch Source specifies what IDO collection is to be used for the search. You can select which IDO properties to include in the DataSearch. You can also specify which IDO properties to display as part of the results of the DataSearch, regardless of whether those properties were included in the search.

For example, you might want to set up a DataSearch Source to search across customer records to identify customers who have recently placed orders for bicycle seats with part numbers prefixed by "CP-".

In this case, you would not want the DataSearch to search through all the IDO properties for the customer name, address, and contact information, because that would extend the search time. However, you would want to display that data for all customers who have placed orders for that particular bicycle seat recently.

You would set up your DataSearch Source to search through orders for which the part number includes the "CP-" prefix and sort them by date, and then retrieve the customer information for those recent orders.

Keep in mind that each DataSearch Source can include data from only one IDO collection. This means that, if you want to search across multiple IDO collections, you will need multiple DataSearch Sources.

DataSearch Source Sets

In cases where you want to include data from several unrelated data sources (IDOs), you can easily create DataSearch Source Sets. DataSearch Source Sets allow you to group several DataSearch Sources into a single set which you can then search all at the same time.

Searching for Application Data 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.

To search for a value:

1. Open the **DataSearch** form.
2. Specify the search value, which can include an asterisk (*) as a wildcard.

   Note:

   - Searches are not case sensitive: The system treats "Test" the same as "test" when querying.
   - The wildcard is not generally needed, because all value interpreters except **`Exact Phrase`** use implied wildcards. The **`Begins with`** value interpreter, for example, treat search terms like this: *Search Term**

   When using **`Any of these words`** or **`All of these words`** as the value interpreter, you can specify multiple values to search for, separating them with spaces. Other value interpreters treat whatever is in the **Search** field as literal values, including spaces, unless the asterisk is used.

3. Specify how to interpret the value:

This value interpreter:	Handles the search term(s) like this:
`Any of these words`	The search retrieves all data with values that have any one of the search terms, similar to a Boolean OR clause.
`All of these words`	The search retrieves only data that has all of the specified search terms, similar to a Boolean AND clause.
`Contains`	The search retrieves only records that contain the search term exactly as specified. For example, a search for the term **`"testuser"`** would retrieve records containing both "testuser" and "testuser1".
`Begins with`	The search retrieves only records that contain the entire search term, exactly as specified, but only when the search term is used at the beginning of the field value in which it is found. For example, a search for the term **`"test user"`** would retrieve a record that contains the value "Test User John", but not for record "John Test User".

This value interpreter:	Handles the search term(s) like this:
`Ends with`	The search retrieves only records that contain the entire search term, exactly as specified, but only when the search term is used at the end of the field value in which it is found. For example, a search for the term `"test user"` would retrieve a record that contains the value "John Test User", but not for record "Test User John".
`Exact Phrase`	The search retrieves records that contain only the search term, exactly as specified. For example a search for the term `"testuser"` would retrieve records containing "testuser" but not "testuser1".

Note: Queries that use the value interpreters `Any of these words` and `All of these words` search across all searchable properties. In these cases, each word is treated as a separate term and searched for individually.

In contrast, the other four value interpreters are variants on phrase-matching, where the whole search expression is treated as one phrase that must be matched within each searchable property.

4 Optionally, specify which data sources to look in.

By default, all data sources are used.

Note: You can define additional custom DataSearch Sources.

5 To perform the search, click the search (magnifying glass) button.

The results are displayed, initially grouped by data source. The **Count** field indicates how many records in the data source include the search term, not how many instances of the search value were located.

Note: If the search term is found in a property that was both searched and displayed, it is highlighted with an amber background. If the search term is found in a property that was searched but not displayed, then nothing in the information that is displayed is highlighted in this way.

6 To expand a data source and view the results in a grid, click the **+** button.

Once the DataSearch query has been done, you have many of the same layout, display, print, and export options as for any other type of DataView display.

There are some differences, however. For one thing, display options are fewer and more limited, basically to expanding or collapsing the results from each DataSearch Source and to rearranging columns.

Also, you can only have a single layout for each DataSearch Source at each level of scope.

Configuring Data Sources for DataSearch

To configure IDOs (sources) to be searched with DataSearch.

1 Open the **DataSearch Source Setup** form and execute Filter In Place.

2 In the **Source Name** field, specify the name by which to identify the DataSearch source.

3 In the **IDO Name** field, specify the IDO to use as the source for the data.

   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).

4 Optionally, set up a filter to use with the DataSearch source.

   To help you set up this filter and to ensure that the filter is formatted properly, you can click **Filter**, which launches the**DataSearch Source Filter Setup** form.

5 Optionally, to provide a name (caption) to override the **Source Name** and be used for display purposes in DataSearch forms, specify that caption in the **Caption Override** field.

   If the value you specify is formatted as a translatable string,for example, "sProduct" rather than "Product," the string is translated. 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.

6 Optionally, use the **Record Cap** field to limit the amount of data that is initially displayed in the search results. Select one of these options:

   - Use System Setting: Use the application's record cap setting.
   - Use Specified Max: Specify a maximum value in the **Record Cap Value** field. This is the default value.
   - Retrieve All: This option can affect system performance if many records are retrieved.

7 Use the **Order By** group of fields to specify the order in which records are to be searched and retrieved:

   a Specify the property to use for sorting.

   b Specify the order (ascending or descending).

   c To create the Order By expression, click **Add**

      The expression is displayed in the **Order By** field.

      Note:

      - Be aware that this clause does not determine the order in which the retrieved results are presented in the DataSearch display. This clause determines the order in which the records are retrieved to begin with. Among other things, this means that, if the number of records queried is greater than the record cap, an "ascending" query does not retrieve the same set of records as a "descending" query.

        For example, searching on a customer name field using an ascending Order By clause might return only the records of customers whose names begin with the letters A-G, whereas searching the same field using a descending Order By clause might return only the records of customers whose names begin with the letters Z-S.

      - Override this Order By clause in the DataSearch results by using a Sort By statement at the DataView layout level.

8 On the **General** tab, select the IDO properties to be searched and the IDO properties to be displayed in the results:

- To select the same properties in the **Properties To Show** grid that are selected in the **Properties To Search** grid, click **Select Search Properties**.
- Optionally, you can then 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 **Properties To Show** results but not in the **Properties To Search** list.
- To include any notes that might be attached to the records being retrieved, select the **Search Notes** check box and/or the **Show Notes** check box.

9 Optionally, on the **Search Sets** tab, specify any DataSearch Source Sets that you want to include in the searches.

10 Optionally, on the **User Permissions** tab, specify the users or groups who can view results from searches performed using this DataSearch source.

11 To launch the **DataSearch** form with this source selected, click **Launch**.

Setting Up a DataSearch Filter

1 To launch the **DataSearch Source Filter Setup** form, click **Filter** on the **DataSearch Source Setup** form.

Note: You cannot open this form directly.

2 In the **Property Name** field, specify the IDO property for which you want to set up the filter.

3 Specify the operator to use for the filter. If you select `IS NULL` or `IS NOT NULL`, the comparison field and value need not be specified.

4 Select the comparison type:

- `Literal`: The property value is compared to the literal value that you specify in the last field on this row.
- `DataSearch Source Property:` The property value is compared to another property value from the DataSearch source.

  This can be useful when you want to use a comparison of two properties to control what records are searched. For example, if you wanted to search only orders that do not have the full number of items shipped, you could set up the property comparison `QtyShipped < QtyOrdered`. Or suppose you wanted to search only orders that were not shipped on time, you could set up the property comparison `ShipDate > DueDate`.

5 In the last field on this row, specify the comparison value to be used for the corresponding comparison type.

6 Optionally, if you use multiple filter clauses, to instruct the system to treat the clause as Boolean OR comparisons rather than Boolean AND comparisons with the previous clause, select **OR Instead of AND with Previous Clause**.

7 To add the filter clause to the list of clauses in the display panel, click **Add**.

The clause is added to the list, formatted with the proper syntax.

8 Optionally, use Steps 2 through 7 to specify additional filter clauses.

9 Optionally, to remove all filter clauses and start over, click **Remove**.

Note: You cannot selectively remove individual filter clauses: If you choose to remove one, you remove them all.

10 Click **OK**.

For complex comparison logic, it might be necessary to add parentheses around filter clauses, to make sure they are evaluated properly. To accomplish this task, manually edit the data in the display list.

Note: To get a better understanding of how filters can be used, look at the vendor level DataSearch Sources provided as part of the application installation.

About DataSearch Source Sets

A DataSearch Source Set allows you to group multiple DataSearch Sources together, to make it easier for users to search multiple sources at one time.

For example, you might want to perform a DataSearch that includes records from the Customers, Customer Orders, Customer Order Lines, and Invoices IDO collections. The easiest way to accomplish this is to create a DataSearch Source for each IDO, and then group the four sources into a DataSearch Source Set named "Customer Info."

To create and maintain DataSearch Source Sets, use the **DataSearch Source Sets** form.

Creating a DataSearch Source Set

To create a DataSearch source set:

1 Open the **DataSearch Source Sets** form.
2 Click in the last row of the grid view to create a new record.
3 In the **Set Name** field, specify the name for the source set.
4 Optionally, in the **Caption Override** field, specify a caption to use when the source set is displayed.
5 In the **Source Name** grid, select each DataSearch source that you want to include in the set.
6 Save your changes.

The DataSearch source set is now available for selection on the **DataSearch** form and the **DataSearch Source Setup** form.

About DataSearch Layouts

Similar to DataView layouts, DataSearch queries can also have saved layouts. Like DataViews, you can rearrange columns, choose to hide or show columns, and hide or show extended captions.

DataSearch layouts also display Source and Count information for each DataSearch Source queried.

However, unlike DataViews, DataSearch queries can have only a single saved layout per DataSearch Source per scope level. Once saved, that layout is applied to the DataSearch that uses that source every time a user of that scope performs the query, at least, until you delete it.

DataSearch source layouts do not support summaries, calculated columns, grouping or filtering.

The search results (that is, the display of Source and Count) also have a layout, so you can choose how to sort the results. 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.

Also like predefined DataView layouts, you can view and maintain DataSearch layouts on the **DataView Layouts** form.

Chapter 11: App Builder/App Hub

About App Builder and the App Hub

App Builder

App Builder is a set of online application design tools that are incorporated in and can be run from within Infor Mongoose. As part of the Infor Technology Suite, it can also run within Infor Ming.le™ and in the cloud as part of a multi-tenant offering.

App Builder allows you to reuse existing business logic across Infor products and to build composite applications that use that business logic. This alows you to create custom-built and highly specialized apps to boost your productivity.

App Builder is designed to build basic apps using an intuitive WYSIWYG user interface. App Builder offers drag-and-drop of UI components, a data service layer to connect with APIs, and an interaction wizard to set up how the components of your app interact with each other.

App Builder also includes a responsive-design mechanism that enables you to easily design apps for optimal operation and display, regardless of what device the end user is using.

Apps can be built, exported, and published from and into different environments. This means you can develop and test in your test environment, and then later deploy in your production environment. You can begin the creation of an app with App Builder and then open it in Mongoose for further development. It also means that you can receive and incorporate apps from other Infor or third-party sources.

Apps built in App Builder are client-side applications that use ION API backend services. Because many Infor products are already ION API-certified, you can use them when building apps with App Builder. It is also possible to manually attach your own APIs behind the ION API gateway, and that way, you can use any REST-based API as part of your apps.

App Hub

The App Hub is a repository where all published and activated App Builder apps are stored. This is the main user interface for the end user of your apps. The App Hub offers a list of all apps the user has access to.

Authorized administrators can administer their companys' apps within the App Hub. This includes control over which apps are available in what versions and who can access them.

Mongoose as host for App Builder and App Hub

When used within Infor Mongoose, App Builder and App Hub are both hosted within Mongoose forms that have the same names. In fact, with the incorporation of App Builder and App Hub into Mongoose, these apps now operate as FormOnly-mode Mongoose forms. This means that, if you are using Ming.le to access App Builder and App Hub, you must be a licensed Mongoose user operating in Ming.le. And if, for some reason, you are attempting to access App Builder or the App Hub from a URL, you are prompted to sign in to Mongoose before the form opens.

Bi-directional design and storage of App Builder apps

App Builder and Mongoose can be considered bi-directional design tools. By this, we mean that what you do in one is automatically synchronized with the other.

For example, suppose you create an app in App Builder. When you do, Mongoose automatically creates and stores the app as a Mongoose form with a special prefix (**AB_**). This means that you can open the App Builder app as a form in Mongoose. What's more, you can further develop the app/form in Mongoose and it automatically becomes part of the original App Builder project.

Note: This does not necessarily imply that all content in one can transfer to the other. There are limitations, in that the metadata from an App Builder project must have an analogous concept or component in Mongoose, or Mongoose does not recognize and use it. Similarly, changes made in Mongoose must align with runtime functionality in App Builder, or they are not recognized or used in App Builder.

The same generally holds true for App Builder projects that are published in the App Hub. When a project is published in the App Hub, it is saved and stored as yet another Mongoose form with a different prefix. This form is separate from the form representing the original App Builder project, so that development can continue on the original App Builder project without disrupting runtime usage in the App Hub.

App Builder interactions and Mongoose form event handlers

Interactions created in App Builder projects are created and stored in their corresponding Mongoose forms as form event handlers. If new event handlers that are applicable in App Builder are then added to the Mongoose form, they are created in App Builder as interactions.

App Builder data services and Mongoose IDOs/form collections

Data services used in App Builder projects are created in Mongoose as IDOs (Intelligent Data Objects). Collections created with/for those IDOs are then added to the Mongoose form, and Mongoose components are bound to those collections.

App Builder permissions, groups, and roles

To use App Builder and the App Hub effectively, you must have the appropriate permissions; be a member of the appropriate user groups; and have the correct roles assigned. Since App Builder and the App Hub are hosted in Mongoose forms, App Builder users must be assigned to the correct license modules, as well as being assigned to appropriate user groups within Mongoose.

Licensing for App Builder and the App Hub

To be authorized to access either App Builder or the App Hub, you must be assigned the MGAppBuilder license module. This license module simply allows you access to those forms.

You must also be assigned the MGUserCreatedForms license module and possibly the MGUserCreated IDOs license module (depending on how involved you want to get on the IDO side of things). These license modules allow you access to any forms you might create using App Builder.

These license assignments are in addition to any other Mongoose license modules you might be assigned.

Group membership for App Builder and the App Hub

Mongoose has three user groups that are predefined for App Builder/App Hub users:

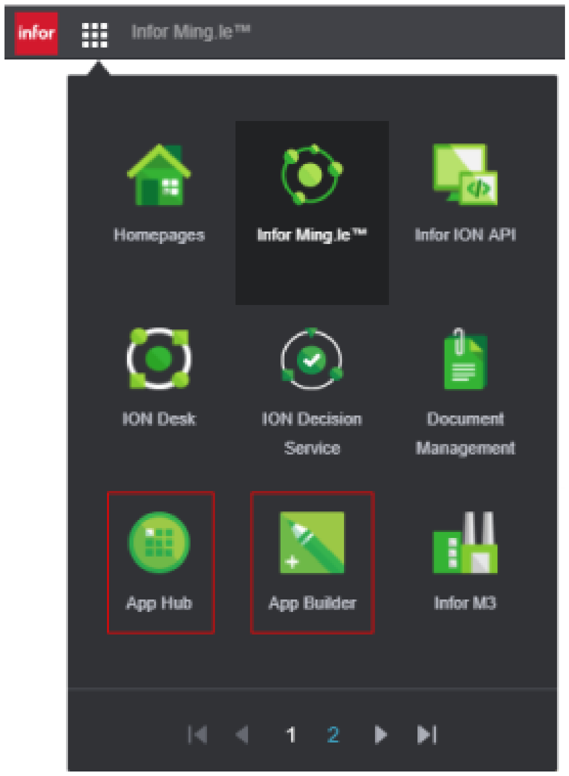
- APPBUILDER-Designer - Members of this group have permission to run and use the App Builder (form). They can only use it to design and create apps. Once their apps are ready, they can publish them to the App Hub (from App Builder). They cannot, however, access the App Hub (form) or activate their apps in the App Hub. That must be done by an Administrator.
- APPBUILDER-User - Members of this group have permission only to access the App Hub (form). They can only use apps built in App Builder, not create or modify them.
- APPBUILDER-Administrator - Members of this group have permissions to run the App Hub. They also have access to additional administrative functionality within the App Hub. For example, they can:
  - Restrict access to apps with IFS/Mongoose roles.
  - Activate or deactivate apps.
  - Upgrade or downgrade apps.
  - Export or import apps.
  - Delete apps from the App Hub.
  - Change or modify names or descriptions of apps.

Users can be assigned to these groups in Mongoose. If Mongoose is running within Ming.le, then the users, user groups (permissions, roles), and user assignments to groups are all synchronized from Ming.le.

Launching App Builder and the App Hub

There are three ways to launch App Builder and the App Hub. You can:

- Launch them in Ming.le, by means of linked icons.



- Launch them in Mongoose, the same as any other Mongoose form.
- Launch them by means of specially structured URLs.

  The URL should follow this pattern:

  ```
  http(s)://serverName:port/WSWebClient/Default.aspx?page=FormOnly&notitle=
  1&Form=AppBuilder
  ```

  where:

  - *serverName* is the name of your web client server, including the domain, if necessary.
  - *port* is the number of the port to use (optional).

Chapter 12: Form Extensions

About extending and replacing forms

Mongoose forms can form an inheritance hierarchy, with child forms extending parent forms in an arbitrary tree structure. This topic introduces a variation of the extension mechanism called “extend and replace.” When utilized, a form that “extends and replaces” a parent form is opened when a request to open the parent form is received, transparent to the user. This is similar to the extend and replace feature for IDOs.

You can enable this extension mechanism by selecting the **Load/replace form with extended form** check box on the **User Preferences** dialog box.

When the user requests a form, IDO Runtime follows this hierarchy of priority to determine which form definition must be returned:

1. **Form Extensions** form entry for **Form Name**

   If the **Form Name** value is the same as the **Replaced By Form** value, the extend and replace feature will not take place. IDO Runtime simply returns the requested form definition.
2. Descendant forms in extension hierarchy with **Replace Base Form** property set to **`True`**
3. Highest descendant level
   - If there is no form found at this level, then the requested form definition is returned.
   - If there is one form found at this level, then the requested form definition is replaced by that descendant form definition.
   - If there are multiple forms found at this level, then you must use the **Form Extensions** form to determine which form definition must be returned.
4. **Form Extensions** form entry:
   a. **Language ID**
   b. **Scope Type**

To identify which extended form is used to replace the requested form based on the hierarchical level, select **Help > About This Form** then click the **Warning Message(s)** button.

Using the Form Extensions form

It is possible that more than one extended form can be created to replace a base form or an extended form on the same system. To manage the details about the extended form that you created and to determine which extended form must be returned when a request to open the base form is received, use the **Form Extensions** form.

Field	Description
Form Name	The name of the base form **Note:** This is the actual form name, and not the form caption.
Scope Type	The scope under which the extended form is created: • Vendor • Site • Group • User
Scope Name	The name of the group or user for which the form is created **Note:** This field is populated only if the form is extended at the Group or User level.
Language ID	The language and a region or country that is used for translation
Replaced By Form	The name of the extended form
Access As	Read-Only. The current **Access As** identifier, which is used in some forms to identify who created the metadata object. Default and other possible values include: • `Core` - Metadata that Infor has created as part of the basic system. • *OtherName* - Metadata created as part of an application. This value is variable and is set by the creator of the application. • Blank - This field should be blank for any Access As-controlled metadata you have created.