



PRODUCTS

- Database
- Middleware
- Developer Tools
- Enterprise Management
- Applications Technology
- Products A-Z

TECHNOLOGIES

- BI & Data Warehousing
- Embedded
- Java
- Linux
- .NET
- PHP
- Security
- Windows Server System
- Technologies A-Z

ARCHITECTURE

- Enterprise 2.0
- Extreme Transaction Processing
- Grid
- Service-Oriented Architecture
- Virtualization

COMMUNITY

- Join OTN
- Oracle ACEs
- Oracle Wiki
- Blogs
- Podcasts
- Events
- Newsletters
- Oracle Magazine
- Oracle Books
- Certification
- User Groups
- Partner White Papers

Oracle JDeveloper Tip

ADF Equivalents of Common Oracle Forms Triggers

Author: Steve Muench, ADF Development Team

Date: May 24, 2005

Revision 1.0 ([Revision History](#))

Abstract

This paper provides a quick summary of how basic tasks performed using the most common Oracle Forms triggers are accomplished using the Oracle ADF framework. If you have other common Forms triggers that you use frequently, drop me an [email](#) and I'll update the paper to include those, too.

Contents

- [Validation & Defaulting \(Business Logic\)](#)
- [Query Processing](#)
- [Database Connection](#)
- [Transaction "Post" Processing \(Record Cache\)](#)
- [Error Handling](#)

Validation & Defaulting (Business Logic)

Forms Trigger	Usage	ADF Equivalent
WHEN-VALIDATE-RECORD	Execute validation code at the record level	In the custom EntityImpl class for your entity object, write a public method returning <code>boolean</code> type with a method name like <code>validateXXXX()</code> and have it return <code>true</code> if the validation succeeds or <code>false</code> if the validation fails. Then, add a method validator for this validation method at the entity level on the "Validation" panel for your entity object. When doing that you can associate a validation failure message with the rule at that time.
WHEN-VALIDATE-ITEM	Execute validation code at the field level	In the custom EntityImpl class for your entity object, write a public method returning <code>boolean</code> type and accepting a single argument of the same datatype as your attribute, having a method name like <code>validateXXXX()</code> . Have it return <code>true</code> if the validation succeeds or <code>false</code> if the validation fails. Then, add a method validator for this validation method at the entity attribute level for the appropriate attribute on the "Validation" panel for your entity object. When doing that you can associate a validation failure message with the rule at that time.
WHEN-DATABASE-RECORD	Execute code when a row in the datablock is marked for INSERT or UPDATE.	Override the <code>addToTransactionManager()</code> method of your entity object. Write code after calling the super.

WHEN-CREATE-RECORD	Execute code to populate complex default values when a new record in the data block is created, without changing the modification status of the record.	Override the <code>create()</code> method of your entity object and after calling <code>super</code> , use appropriate <code>setAttrName()</code> methods to set default values for attributes as necessary. To eagerly set a primary key attribute to the value of a sequence, construct an instance of the <code>SequenceImpl</code> helper class and call its <code>getSequenceNumber()</code> method to get the next sequence number. Assign this value to your primary key attribute. If you want to assign the sequence number lazily, but still without using a database trigger, you can use the technique above in an overridden <code>prepareForDML()</code> method in your entity object. If instead you want to assign the primary key from a sequence using your own <code>BEFOREINSERTFOREACHROW</code> database trigger, then use the special datatype called <code>DBSequence</code> for your primary key attribute instead of the regular <code>Number</code> type.
WHEN-REMOVE-RECORD	Execute code whenever a row is removed from the data block.	Override the <code>remove()</code> method of your entity object and write code either before or after calling <code>super</code> .

Query Processing

Forms Trigger	Usage	ADF Equivalent
PRE-QUERY	Execute logic before executing a query in a Data Block, typically to setup values for query-by-example criteria in the "example record".	Override <code>executeQueryForCollection()</code> on your view object class and write code before calling the <code>super</code> .
ON-COUNT	Override default behavior to count the query hits for a Data Block	Override <code>getQueryHitCount()</code> in your view object and do something instead of calling the <code>super</code> .
POST-QUERY	Execute logic after retrieving each row from the datasource for a data block.	Generally instead of using a <code>POST-QUERY</code> style technique to fetch descriptions from other tables based on foreign key values in the current row, in ADF it's more efficient to build a view object that has multiple participating entity objects, joining in all the information you need in the query from the main table, as well as any auxiliary/lookup-value tables. This way, in a single round-trip to the database you get all the information you need. If you still need a per-fetched-row trigger like <code>POST-QUERY</code> , override the <code>createInstanceFromResultSet()</code> method in your view object class.
ON-LOCK	Override default behavior to attempt to acquire lock on the current row in the data block.	Override the <code>lock()</code> method in your entity object class and do something instead of calling the <code>super</code> .

Database Connection

Forms Trigger	Usage	ADF Equivalent
POST-LOGON	Execute logic after logging onto the Database	Override <code>afterConnect()</code> on your custom application module. Since application module instances can stay connected while serving different logical client sessions, probably what you want is to override the <code>prepareSession()</code> which is fired after initial login, as well as after any time the application module is used by a user that was different from the one that used it last time.
PRE-LOGOUT	Execute logic before logging off from the Database	Override <code>beforeDisconnect()</code> on your custom application module class and write code.

Transaction "Post" Processing (Record Cache)


Forms Trigger	Usage	ADF Equivalent
PRE-COMMIT	Execute code before commencing processing of the changed rows in all data blocks in the transaction.	Override <code>commit()</code> method in a custom <code>DBTransactionImpl</code> class and write code before calling the super. Note: See this article for an overview of creating and using a custom <code>DBTransaction</code> implementation.
PRE-INSERT	Execute code before NEW row in the datablock is INSERTed into the database during "post" processing.	Override <code>doDML()</code> method in your entity class and if the <code>operation</code> equals <code>DML_INSERT</code> then write code <i>before</i> calling the super.
ON-INSERT	Override default processing for INSERTing a NEW row into the database during "post" processing.	Override <code>doDML()</code> method in your entity class and if the <code>operation</code> equals <code>DML_INSERT</code> then write code <i>instead of</i> calling the super.
POST-INSERT	Execute code after NEW row in the datablock is INSERTed into the database during "post" processing.	Override <code>doDML()</code> method in your entity class and if the <code>operation</code> equals <code>DML_INSERT</code> then write code <i>after</i> calling the super.
PRE-DELETE	Execute code before row removed from the datablock is DELETED from the database during "post" processing.	Override <code>doDML()</code> method in your entity class and if the <code>operation</code> equals <code>DML_DELETE</code> then write code <i>before</i> calling the super.
ON-DELETE	Override default processing for DELETing a row removed from the datablock from the database during "post" processing.	Override <code>doDML()</code> method in your entity class and if the <code>operation</code> equals <code>DML_DELETE</code> then write code <i>instead of</i> calling the super.
POST-DELETE	Execute code after row removed from the datablock is DELETED from the database during "post" processing.	Override <code>doDML()</code> method in your entity class and if the <code>operation</code> equals <code>DML_DELETE</code> then write code <i>after</i> calling the super.
PRE-UPDATE	Execute code before row changed in the datablock is UPDATED in the database during "post" processing.	Override <code>doDML()</code> method in your entity class and if the <code>operation</code> equals <code>DML_UPDATE</code> then write code <i>before</i> calling the super.
ON-UPDATE	Override default processing for UPDATing a row changed in the datablock from the database during "post" processing.	Override <code>doDML()</code> method in your entity class and if the <code>operation</code> equals <code>DML_UPDATE</code> then write code <i>instead of</i> calling the super.
POST-UPDATE	Execute code after row changed in the datablock is UPDATED in the database during "post" processing.	Override <code>doDML()</code> method in your entity class and if the <code>operation</code> equals <code>DML_UPDATE</code> then write code <i>after</i> calling the super.
POST-FORMS-COMMIT	Execute code after Forms has "posted" all necessary rows to the database, but before issuing the data COMMIT to end the transaction.	If you want a single block of code for the whole transaction, you can override the <code>doCommit()</code> method in a custom <code>DBTransactionImpl</code> object and write some code before calling <code>super</code> . To execute entity-specific code before commit for each affected entity in the transaction, override the <code>beforeCommit()</code> method on your entity object and write some code there.
POST-DATABASE-COMMIT	Execute code after database transaction has been committed.	Override <code>commit()</code> method in a custom <code>DBTransactionImpl</code> class and write code <i>after</i> calling the super.

Error Handling

Forms Trigger	Usage	ADF Equivalent
ON-ERROR	Override default behavior for handling an error.	Install a custom error handler (<code>DCErrorHandler</code>) on the ADF <code>BindingContext</code>

Revision History

Date	Comments
24-May-2005	Created

 [E-mail this page](#)

 [Printer View](#)