

# CA Gen

## Action Diagram User Guide

Release 8.5



Second Edition

This Documentation, which includes embedded help systems and electronically distributed materials, (hereinafter referred to as the "Documentation") is for your informational purposes only and is subject to change or withdrawal by CA at any time.

This Documentation may not be copied, transferred, reproduced, disclosed, modified or duplicated, in whole or in part, without the prior written consent of CA. This Documentation is confidential and proprietary information of CA and may not be disclosed by you or used for any purpose other than as may be permitted in (i) a separate agreement between you and CA governing your use of the CA software to which the Documentation relates; or (ii) a separate confidentiality agreement between you and CA.

Notwithstanding the foregoing, if you are a licensed user of the software product(s) addressed in the Documentation, you may print or otherwise make available a reasonable number of copies of the Documentation for internal use by you and your employees in connection with that software, provided that all CA copyright notices and legends are affixed to each reproduced copy.

The right to print or otherwise make available copies of the Documentation is limited to the period during which the applicable license for such software remains in full force and effect. Should the license terminate for any reason, it is your responsibility to certify in writing to CA that all copies and partial copies of the Documentation have been returned to CA or destroyed.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CA PROVIDES THIS DOCUMENTATION "AS IS" WITHOUT WARRANTY OF ANY KIND, INCLUDING WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NONINFRINGEMENT. IN NO EVENT WILL CA BE LIABLE TO YOU OR ANY THIRD PARTY FOR ANY LOSS OR DAMAGE, DIRECT OR INDIRECT, FROM THE USE OF THIS DOCUMENTATION, INCLUDING WITHOUT LIMITATION, LOST PROFITS, LOST INVESTMENT, BUSINESS INTERRUPTION, GOODWILL, OR LOST DATA, EVEN IF CA IS EXPRESSLY ADVISED IN ADVANCE OF THE POSSIBILITY OF SUCH LOSS OR DAMAGE.

The use of any software product referenced in the Documentation is governed by the applicable license agreement and such license agreement is not modified in any way by the terms of this notice.

The manufacturer of this Documentation is CA.

Provided with "Restricted Rights." Use, duplication or disclosure by the United States Government is subject to the restrictions set forth in FAR Sections 12.212, 52.227-14, and 52.227-19(c)(1) - (2) and DFARS Section 252.227-7014(b)(3), as applicable, or their successors.

Copyright © 2014 CA. All rights reserved. All trademarks, trade names, service marks, and logos referenced herein belong to their respective companies.

## CA Technologies Product References

This document references the following CA Technologies products:

- CA Gen
- AllFusion® Gen

## Contact CA Technologies

### Contact CA Support

For your convenience, CA Technologies provides one site where you can access the information that you need for your Home Office, Small Business, and Enterprise CA Technologies products. At <http://ca.com/support>, you can access the following resources:

- Online and telephone contact information for technical assistance and customer services
- Information about user communities and forums
- Product and documentation downloads
- CA Support policies and guidelines
- Other helpful resources appropriate for your product

### Providing Feedback About Product Documentation

If you have comments or questions about CA Technologies product documentation, you can send a message to [techpubs@ca.com](mailto:techpubs@ca.com).

To provide feedback about CA Technologies product documentation, complete our short customer survey which is available on the CA Support website at <http://ca.com/docs>.

## Documentation Changes

The following documentation updates have been made since the last release of this documentation:

- [Call External Statement](#) (see page 59)—Describes the CALL EXTERNAL statement.
- [Inline Code Statement](#) (see page 58)—Describes the Inline Code statement.

# Contents

---

## Chapter 1: Introduction 9

Process Action Diagrams .....	9
Procedure Action Diagrams.....	10

## Chapter 2: Building the Action Diagram 11

Views .....	11
View Maintenance .....	12
Action Diagram View Maintenance .....	12
How to Define Logic for Elementary Processes and Procedures .....	13
Define Process as Elementary in Activity Model .....	13
Add Entity Actions.....	14
Add Relationship Actions .....	31
Add Assignment Actions .....	33
Add Conditional Actions.....	59
Add Repeating Actions .....	64
Add Control Actions .....	73
Add Miscellaneous Actions .....	76
Define Action Blocks.....	78
Create New Action Block in Analysis or Design.....	78
Define Action Blocks for Processes and Procedures .....	78
Reference Common Action Blocks .....	79
Define External Action Blocks .....	80
Changing the Action Diagram.....	80
Complex Changes .....	80
Perform Simple Changes.....	81
Copying an Action Diagram .....	81

## Chapter 3: Event Processing and GUI Statements 83

Adding Event Actions .....	83
Event Action Names .....	83
Event Types .....	84
CA Gen Supplied Event Types .....	84
User-Defined Event Types .....	85
Action Diagram Statements for GUI Applications .....	86
GUI Object Domain .....	87
Dot notation.....	87

---

## Chapter 4: Introduction to Views 89

Views .....	89
Using Views .....	90

## Chapter 5: Creating Views 93

Common Steps for Adding Views .....	93
How to Create Views in the Import View Subset .....	94
Create Import Entity Views .....	97
Create Import Work Views .....	99
Create Views in the Export View Subset .....	101
Create Views in the Local View Subset .....	103
Create Views in the Entity Action View Subset .....	104
Add Entity Action Views .....	104
Detail Entity Action Views .....	105
Assign View Characteristics .....	105
Supports Entity Actions .....	105
Lock Required on Entry .....	107
Used as Both Input and Output .....	108
Recommendations .....	109
How to Change Views .....	109
Change Parent of Views .....	110
Copy Views .....	110
Delete Views .....	110
Move Views .....	111
Rename a View .....	111

## Chapter 6: Using Views 113

Associate Views with External Objects (Add) .....	113
Synthesize Views in the DLG .....	113
Match Views .....	115
Guidelines for Matching Views .....	116
Match Views Using the PAD .....	117
Match and Copy Views .....	120
Unmatching Views .....	120
Maximum Size of Views .....	121

## Chapter 7: Process Synthesis 123

When to Use Process Synthesis .....	123
Process Synthesis Output .....	124

---

Prerequisites .....	126
How to Access Process Synthesis .....	127
Process Synthesis from the Data Model .....	128
Process Synthesis from the Action Diagram .....	128
Process Synthesis from Activity Hierarchy and Activity Dependency .....	128

## **Chapter 8: Using Process Synthesis** **129**

How to Create a Process Logic Diagram .....	129
Select an Elementary Process to Analyze .....	130
Identify Entity Types Used in Process .....	130
Identify Required Actions .....	131
Identify Sequence of Actions .....	131
Perform Process Synthesis .....	132
Process Synthesis Rules .....	133
Process Synthesis from Data Model .....	136
Process Synthesis from Activity Hierarchy and Activity Dependency .....	138
Process Synthesis from Action Diagram .....	138
Review Results of Process Synthesis .....	139
Review Generated Expected Effects .....	140
Review Generated Views .....	140
Review Generated Action Diagrams .....	141

## **Chapter 9: Structure Chart and Action Block Usage Tools** **149**

Structure Chart .....	150
How to Access the Structure Chart .....	151
Action Block Usage Tool .....	151
Access the Action Block Usage Tool .....	152
Using a Structure Chart or Action Block Usage Diagram .....	153
Add an Action Block .....	153
Detail an Action Block .....	153
Using Other Action Options .....	154
Chain .....	154
Check .....	155
Contract .....	155
Expand .....	155
Expand All .....	155
Redraw .....	155
Unmatch .....	156

---

## Chapter 10: Procedure Synthesis 157

Create a Procedure Action Diagram .....	157
Prerequisites .....	158
How to Access Procedure Synthesis .....	158

## Chapter 11: Using Procedure Synthesis 159

Select the Stereotype .....	159
Entity Maintenance Stereotype .....	160
Selection List Stereotype .....	161
Implementation of Action Blocks Stereotype .....	161
Select Stereotype Options .....	162
Add an Action Bar for Command Entry .....	162
Execute Entity Actions in One or Two Stages .....	164
Use the Enter Command to Update .....	166
Use the Enter Command to Display .....	166
Refresh the Screen on Cancel .....	166
Clear Data from the Screen .....	167
Synthesize Flows .....	168
Synthesize Menu Flows .....	168
Synthesize Subject List Flows .....	169
Synthesize Neighbor Selection List Flows .....	169
Resynthesize Processes .....	170

## Chapter 12: Asynchronous Behavior 171

ASYNC_REQUEST Work Set .....	171
Async Action Statements .....	172
USE ASYNC .....	172
GET ASYNC RESPONSE .....	178
CHECK ASYNC RESPONSE .....	181
IGNORE ASYNC .....	183

## Index 185



# Chapter 1: Introduction

---

Interaction analysis identifies the effects that processes have on entities, attributes, and relationships, found in CA Gen.

The Action Diagram tool lets you build the Process Action Diagram, Procedure Action Diagram, and the associated action blocks. These specify the detailed logic of elementary processes, procedure steps, business algorithms, and derivation algorithms.

The first part of an Action Diagram details the views that the process imports and exports. The remaining pseudocode consists of action statements such as CREATE, READ, UPDATE, DELETE, and so on, with levels of nesting enclosed in brackets. These statements include the actual effects that a process has at execution time. The statements provide the detailed process logic on which code generation is based.

**More information:**

[Introduction to Views](#) (see page 89)

## Process Action Diagrams

The product of interaction analysis is the Process Action Diagram that consists of an ordered collection of actions that defines an elementary process.

You can also generate statements that appear in a Process Action Diagram using tools other than the Action Diagram. For example, you can expand the expected effects of processes defined with the Activity Hierarchy Diagram or Matrix Processor into actual effects and views in a Process Action Diagram. Similarly, when you detail views, the software generates corresponding entries in a Process Action Diagram.

Process logic analysis identifies actions to be performed on entities by an elementary process. The actions produce output information based on input information. Process logic analysis also deals with how entity actions act on data in the underlying Data Model. Procedure logic builds on the process logic to identify how the system interacts with the end user.

The following list details the objectives of process logic analysis:

- Define each elementary process rigorously and unambiguously by detailing its actions upon entities, relationships, and attributes
- Ensure that the Data Model can support all the processes

- Combine the results of data analysis and activity analysis so that it is clear how changes to the process definition affect data analysis
- Provide the data designer with the basic information necessary to define data structures
- Provide the starting point for transformation to Design and the definition of procedures

## Procedure Action Diagrams

The extension of interaction analysis into Design results in the Procedure Action Diagram, which consists of an ordered collection of actions that defines a Procedure Step. A Procedure Step is a useful subdivision of a procedure that performs a discrete and definable amount of processing necessary to complete a procedure. A procedure is a method by which one or more elementary processes are carried out using a specific implementation technique.

The Procedure Action Diagram defines the detailed logic of a Procedure Step. It allows the designer to define the following fields:

- Fields local to the Procedure Step
- Fields required to process the screen that interacts with the Procedure Step
- Fields required to process the database (also available in the Process Action Diagram)
- Exit states or conditions that terminate processing, also available in Process Action Diagram

The following list details the objectives of the procedure logic design:

- Express the business system design at a level that includes data access, important conditional actions, reference to fields in screens, and the use of algorithms
- Provide a basis for generating the business system

The Procedure Action Diagram complements Dialog Design. While the Dialog Flow Diagram defines the logic between Procedure Steps, the Procedure Action Diagram defines the detailed logic within each step, including setting the exit states that control the flows to and from each step.

The Procedure Action Diagram also complements Screen Design and Window Design functionality, tools that control the user interface. Statements in a Procedure Step can set the properties of fields, dialog boxes, and windows, and control event processing activities.

# Chapter 2: Building the Action Diagram

---

Building an Action Diagram requires three main activities:

- Creating views
- Defining logic for elementary processes and procedures
- Defining action blocks

After an Action Diagram is built, it can be changed to fit the business need.

**More information:**

[Changing the Action Diagram](#) (see page 80)

## Views

A view is a collection of associated attributes input to or output from a business activity. Views are the way processes see entities and are provided on a need-to-know basis. For example, one process to view only the Status attribute of an entity occurrence of CUSTOMER to accomplish its task. Another process to see the Name and Address attributes of CUSTOMER. Define the scope of views carefully, you control and restrict access to shared information.

The creation and usage of views are integral model-building activities and can be performed in several tools. Views are not tool-dependent.

**Downstream Effects:** During the code generation stage of Construction, you generate a system that consists of screens or windows populated with the fields you initially defined with views. CA Gen bases the generated program logic on the manipulation of views in the Procedure Action Diagrams.

**More information:**

[Creating Views](#) (see page 93)

[Using Views](#) (see page 113)

## View Maintenance

View maintenance allows you to update, revise, add to, or edit a view set, which is a collection of views for a process. You use view maintenance to match views between processes and the action blocks they call and between Procedure Steps and the action blocks they call. The import view subset defines the information input to a process or action block. The export view subset defines the information produced (exported) by the process or action block. The entity action view subset defines stored information about an entity that a process manipulates during execution; views in this subset are the object of the entity actions: CREATE, READ, UPDATE, DELETE, and READ EACH statements. The local view subset defines information from an entity used solely within a process, not exported. If needed, you can add missing views to each subset.

The Process Action Diagram automatically reflects the import and export view subsets you defined with the Activity Hierarchy Diagram and Activity Dependency Diagram. The Procedure Action Diagram uses synthesized import and export views from elementary processes, views from Analysis and Design action blocks that Process Action Diagram calls, and any new views such as local views.

### More information:

[Using Views](#) (see page 113)

## Action Diagram View Maintenance

Action Diagram view maintenance differs from view maintenance in one aspect—you work directly with views in an Action Diagram or action block without accessing view maintenance from the Detail action. You can add new views, delete views, if they are not used in action statements, copy views, and change details of group, entity, work, and attribute views being imported. You can also change the way the views display on the Action Diagram view list.

You can modify specific details about views directly in the Action Diagram view list. These details include the optionality of entity, work, and attribute views being imported; and the optionality, cardinality, and indexing of group views being imported. These details appear in parentheses next to view names when you expand the list.

Action Diagram view list information examples are as follows:

### Example 1

This example illustrates an optional import entity view that supports entity actions, cannot be used for both import and export, and is unlocked upon entry:

entity input customer (optional, persistent, import only, unlocked)

### Example 2

This example illustrates a repeating group view that is mandatory, has a maximum cardinality of 6, is explicitly indexed, and can be used for both import and export:

group (r) in\_group product information (mandatory, 6, explicit, exported)

You can add views to the view lists in Action Diagrams and action blocks. Also, you can add new attributes or existing attributes to entity views and work views.

#### More information:

[Changing the Action Diagram](#) (see page 80)

[Copying an Action Diagram](#) (see page 81)

[Using Views](#) (see page 113)

## How to Define Logic for Elementary Processes and Procedures

The second part of building an Action Diagram is to define logic for elementary processes and procedures, consists of the following activities:

1. Define process as elementary in activity model
2. Add entity actions
3. Add relationship actions
4. Add assignment actions
5. Add conditional actions
6. Add repeating actions
7. Add control actions
8. Add miscellaneous actions

### Define Process as Elementary in Activity Model

The decomposition of activities in the Activity Hierarchy Diagramming portion of Analysis results in lowest-level processes named *elementary processes*. An elementary process is the smallest self-contained unit of business activity of meaning to an end user. When complete, an elementary process leaves the business in a consistent state. Elementary processes interrogate or change the state of the business. Add Product, Create Customer, and Process Order are examples of elementary processes.

The starting point for defining logic for elementary processes is to specify that a process is either a Process Hierarchy Diagram (PHD) or a Process Dependency Diagram (PDD) is elementary. When you define a process as elementary, CA Gen automatically creates a Process Action Diagram for that process with the same name. The Action Diagram is a shell to which you add logic that summarizes how the elementary process views entity types. You can also expand the expected effects of the elementary process on entity types.

## Add Entity Actions

An entity action is an action performed on entity action views to retrieve and manipulate information about entities. Entity actions are CREATE, READ, READ EACH, UPDATE, and DELETE. To perform an entity action, the entity action views must exist. The following table shows examples of entity actions:

Entity Action	View Name	Entity Type
CREATE	new	customer
READ	required	order
READ EACH	existing	customer
UPDATE	required	part
DELETE	cancelled	order

**Note:** For more information about an entity action, see the *Toolset Help*.

You can use one or more of the following activities to add entity actions:

- Generate elementary processes
- Expand expected effects in the Action Diagram
- Directly add entity actions while working with the Action Diagram

These activities establish the framework for the subsequent development of process and procedure logic. All other logic in the Action Diagram is built around the entity actions you first establish.

## Generate Elementary Processes

One way to add entity actions to the Action Diagram is to generate elementary processes using Process Synthesis. This activity creates views, expected effects, definition properties, and action statements in Process Action Diagrams and action blocks.

During Process Synthesis, you are prompted to specify a subject entity type against which to generate a process and associated logic. You can also specify that you want Process Synthesis to generate logic for any entity types related to the subject entity type.

For example, in addition to creating Customer, the generated Action Diagram for Create Customer might also include the logic for creating one or more Orders and associating them with Customer. This would be the case if the Data Model included the relationship Each Customer always places one or more Orders.

Similarly, if each Customer is either a Foreign Customer or Domestic Customer, CA Gen might also include the logic to check the Location attribute and update either Foreign Customer or Domestic Customer based on the result. By tracing all the relationships surrounding the entity type selected for Process Synthesis, CA Gen can automatically generate complex logic related to other entity types in the neighborhood of the subject entity type.

Process Synthesis generates the following actions, based on expected effects defined for an entity type and references to other entity types:

- CREATE with MOVE, SET, and ASSOCIATE actions and exception logic
- READ with MOVE, selection conditions, and exception logic
- READ EACH with MOVE and selection conditions
- UPDATE with SET and ASSOCIATE actions
- DELETE with READ and MOVE actions and exception logic

In Analysis, Process Synthesis is available in the Data Modeling tools, the Activity Hierarchy Diagram, the Activity Dependency Diagram, and the Action Diagram. In Design, Process Synthesis is available when you access an Action Diagram that does not contain action statements.

Process Synthesis can save you much time and effort in action diagramming. The generated Action Diagrams contain everything needed for elementary processes except the relevant business rules, which you must add.

**More information:**

[Using Process Synthesis](#) (see page 129)

[Process Synthesis](#) (see page 123)

## Expand Expected Effects in the Action Diagram

Expanding expected effects consists of the following activities:

- Select an elementary process in the Activity Hierarchy Diagram or Activity Dependency Diagram.
- Specify the expected effects of a process on one or more entities.
- Access a Process Action Diagram that contains no existing actions.
- Expand expected effects.

When you expand expected effects, CA Gen generates the following information in the Process Action Diagram:

1. Defined import and export views are added. This is not a specific result of expanding expected effects; it occurs as a result of defining import and export views for a process.
2. Entity action views reflecting the entities you specified for expected effects are added to the Action Diagram. These views are unnamed; you must detail the entity action views to name them.
3. Attribute views reflecting the attributes of the entities you specified for expected effects are added to the entity action views.
4. Any combination of the entity actions (CREATE, READ, UPDATE, or DELETE) on associated entity action views are added. These actions reflect the expected effects you specified for the selected process in Matrices, the Activity Hierarchy Diagram, or the Activity Dependency Diagram. The actions also represent the actual effects of processes on entities.

For example, if you specify the expected effects of READ and DELETE on the entity CUSTOMER in the Activity Hierarchy Diagram, CA Gen populates the entity action view subset with an unnamed view of CUSTOMER. CA Gen also automatically generates a READ customer action and DELETE customer action with appropriate subordinate actions based on other menu selections you make while expanding expected effects. This is an efficient way to populate entity action views in the Action Diagram.

Expanding expected effects applies only to the Process Action Diagram. It is not applicable in Analysis action blocks, Design action blocks, or the Procedure Action Diagram.

## Create Entity Actions in the Action Diagram

Directly adding entity actions in the Action Diagram consists of the following main activities:

- Add CREATE entity action
- Add READ entity action



- Add UPDATE entity action
- Add DELETE entity action
- Add READ EACH entity action

**Note:** For more information, see the *Toolset Help*.

## Add CREATE Entity Action

Adding a CREATE entity action is part of creating entity actions in the Action Diagram. In addition to the CREATE entity action, the following list contains other actions you can specify when creating entity actions:

- Add READ actions for any associated entity types
- SET attributes
- Add any ASSOCIATE actions
- Add exception logic to a CREATE entity action

The CREATE entity action records information about entities once they are of interest to the business. CREATE establishes an entity occurrence. To CREATE an entity occurrence, you must have an entity action view for the entity type. When you CREATE an entity, you must assign values to the identifying attributes. All basic and designed attributes should have either a value explicitly SET or a default value defined.

A complete CREATE action statement is as follows:

```
CREATE entity-view-1
[SET attribute-view-1 TO expression ]...
[ASSOCIATE WITH entity-view-2 WHICH relationship IT]...
WHEN SUCCESSFUL
action-statement-list
WHEN ALREADY EXISTS
action-statement-list
```

When you add a CREATE action, CA Gen gives you the choice to automatically generate SET actions. A SET action assigns a value to an individual attribute view. You have the following options when adding the CREATE entity action:

- SET mandatory attributes
- SET all attributes
- Add no SET statements

When you set all attributes, CA Gen generates SET statements for mandatory and optional attributes. SET statements are not generated for auto number or derived attributes.

Consistency check will detect and report if the mandatory attribute is not set.

CA Gen also generates the ASSOCIATE actions for all identifying relationships. ASSOCIATE establishes a pairing along a relationship between two entities. You can ASSOCIATE entities when the relationship is optional; you must ASSOCIATE them (before the end of the elementary process) when the relationship is mandatory.

If you elect not to SET attributes and generate ASSOCIATE statements for the CREATE action, you must add SET statements that assign a value to all mandatory attributes and any other attributes. You must also add ASSOCIATE statements to allow pairings to be established for identifying relationships.

A complete CREATE action statement with SET and ASSOCIATE statements is shown in this example.

```
CREATE received product
SET code to "RECD"
ASSOCIATE WITH processing warehouse WHICH holds IT
```

In the READ Action Statement example below, the logic creates an ORDER after a READ action verifies the prior existence of a CUSTOMER. In this example, the create only occurs if the READ of CUSTOMER was successful (that is, if a CUSTOMER with the correct value for its name attribute already exists). The business algorithm order number calculation is assumed to return the value of the next available ORDER Number. In the ASSOCIATE clause, CUSTOMER is the unnamed entity action view into which information about the requesting CUSTOMER entity is placed as a result of the READ action.

```
READ customer
WHERE DESIRED customer name IS EQUAL TO requesting
customer name
WHEN successful
CREATE order
SET number USING order number calculation
ASSOCIATE number with customer WHICH places IT
```

### More information:

[Add Relationship Actions](#) (see page 31)

[Add Assignment Actions](#) (see page 33)

## Add Exception Logic to a CREATE Entity Action

Exception logic defines the actions to take if an exception to an entity action occurs. You can add the following exception conditions to a CREATE action:

- WHEN already exists
- WHEN successful
- WHEN permitted value violation

The WHEN already exists condition lets you define the actions to be taken if the entity occurrence already exists. The WHEN successful condition lets you define the actions to be taken when the entity occurrence is successfully created. The WHEN permitted value violation condition lets you manage the error that occurs when the application attempts to write a column to the database with a value other than a permitted value. If the exception condition is not included and a permitted value error occurs, the transaction is terminated. CA Gen-generated applications cannot write data that violates permitted value constraints to the database.

The exception logic is added automatically to the CREATE entity action group. If actions are to be taken when an exception occurs, you must add the exception action to the entity action group. If you do not add logic for the WHEN permitted value violation condition, a runtime error is issued when the condition occurs.

In the following example, the CREATE statement shows exception logic noting that information about entity occurrence (PRODUCT) is already stored.

```
CREATE received product
SET code TO import product_code
ASSOCIATE WITH processing warehouse WHICH holds IT
WHEN successful
WHEN already exists
WHEN permitted value violation
```

## Add READ Entity Action

Adding a READ entity action consists of the following subordinate activities:

- Add READ action
- Specify selection conditions
- Add READ exception logic
- Detail properties of a READ statement

The READ entity action retrieves a single occurrence or multiple occurrences of one or more entity types that satisfy the selection criteria that you specify. The READ entity action makes one or more entity occurrences available for further action.

The READ action statement takes the following form:

```
READ entity-view-list
[WHERE selection-condition]
[WHEN successful
action-statement-list-1]
[WHEN not found
action-statement-list-2]
```

The first line indicates the READ action and the entity action view(s) being read. The second line indicates the selection criteria for the READ. The remaining lines indicate the two types of exception statements used in the READ and statement lists specifying actions to take if the exception occurs.

**Downstream Effects:** To reduce the use of cursors in the DBMS, CA Gen generates a standalone SQL SELECT statement for Action Diagram READ statements that: select only one row, are not referenced by UPDATE, ASSOCIATE, DISASSOCIATE, or DELETE statements, or do not read persistent views.

**Note:** For more information about the code generated for a READ statement, see the *Toolset Help*.

**More information:**

[Creating Views](#) (see page 93)

## Add READ Action

Adding the READ action is the first step in building READ action statements. You can read one or more entities with one READ action. Specifying multiple entities (or entity action views) in the entity-view-list of the READ statement:

- Reduces the amount of text in the Action Diagram.
- Improves the efficiency of the generated program by reducing the number of accesses to the database management system (DBMS). It decreases the number of SQL statements.

However, be careful when using multiple entities in READ statements as it may be necessary to generate SQL statements that are joins when multiple tables are read.

Cartesian joins, results in reading all rows of two or more tables, can seriously impact response time. The generated logic will take maximum advantage of denormalized data to attempt to avoid or reduce joins in generated logic.

**Downstream Effects:** The second benefit takes advantage of denormalization, or replication of data in multiple places. Denormalization is available in the Data Structure Diagram in technical design.

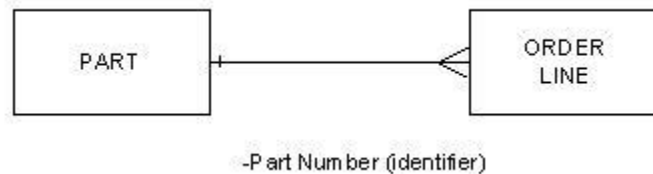
For example, consider a Data Model that contains two entity types, PART and ORDER LINE. PART contains the identifying attribute, Part Number. In the Data Structure Diagram, the relationship between PART and ORDER LINE is implemented by Part Number, which becomes a foreign key. In the database, Part Number is implemented in the PART table as a primary key. Part Number is implemented in the ORDER LINE table as a foreign key. Part Number becomes a denormalized field and is stored redundantly.

Carried one step further, the Action Diagram contains a READ statement that READs both PART and ORDER LINE and contains selection conditions relating PART to ORDER LINE. The resulting benefit occurs during Code Generation. The Code Generation Tool verifies that all fields it requires from PART (in this example, Part Number) are actually stored in ORDER LINE. The tool reads Part Number out of the ORDER LINE table rather than perform a join to the PART table, thereby accessing the database only once. Without the advantages of denormalization and the ability to read multiple views in a single READ statement, both PART and ORDER LINE would have appeared in separate READ statements. A join of the two tables representing each entity type in the database would be necessary to collect data from both.

A READ statement which reads multiple views should always contain a WHERE clause that connects the views in the READ list (relationship or attribute comparison). For example, if the relationship between PART and ORDER LINE were optional from ORDER LINE, you would not place both entity views on the READ list unless you only wanted to see those ORDER LINES that had a pairing to PART.

READ EACH repeating actions, also let you read multiple views. This diagram illustrates a Data Model Representation of Entity Types:

- Specify attribute conditions
- Specify relationship conditions
- Specify system conditions



The first two activities also pertain to the READ EACH repeating action, so references to this appear where appropriate.

The format for READ selection conditions and their placement in the READ statement follows. Expanded views of the READ statement in the following sections show the valid types and combinations of expressions and operandi.

```

READ entity-view-list
WHERE [(] attribute-condition-1 [)] AND [(] attribute-condition-2 [)]
relationship-condition-1 OR relationship-condition-2 ...
system condition-1 system condition-2
[WHEN successful
action-statement-list-1]
[WHEN not found
action-statement-list-2]
  
```

When you specify a combination of attributes and relationships as selection criteria for a READ entity action, specify the attributes first.

The selection conditions that qualify the READ action appear after the WHERE clause. You can combine and join the selection conditions with ANDs, ORs, and parentheses. The selection conditions test attribute values and the existence of a pairing.

You qualify each view being read by at least one selection condition somewhere in the READ statement. Otherwise, all occurrences are read. Your READ statement can be simple or complex. For either simple or complex statements, CA Gen makes statement construction easy by presenting only valid choices.

Use local attributes instead of work set attributes in READ selection criteria. Work set attributes in READ statements might cause poor performance because a domain mismatch can cause the DBMS not to use the available index for searching.

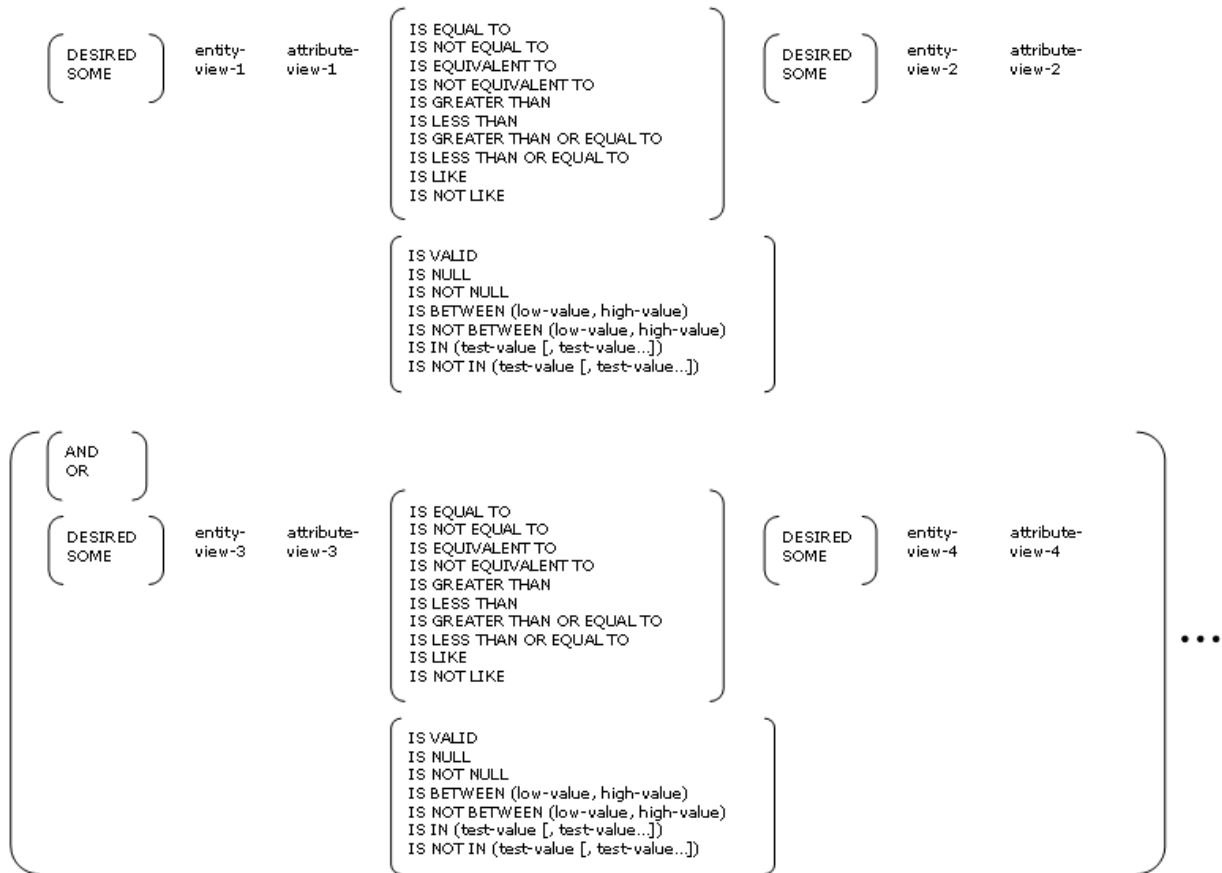
Import entity views that are persistent, import only (locked or unlocked) can be used only as the object of a CURRENT clause in a WHERE clause in a READ or READ EACH statement. The import only designation prevents import views from being used for entity actions.

Export, Persistent, Export only (Locked or Unlocked) can also be used as the object of a CURRENT clause. However, as nothing was passed down from the using Action Diagram, they must first be READ into (or CREATED) before being used in a CURRENT clause. The results can then be exported.

### Specify Attribute Conditions for a READ Action

The first task in specifying selection conditions for a READ (and a READ EACH) action is to add attribute conditions.

The following READ Action Statement shows the attribute condition format in a READ (and READ EACH) statement.



An attribute-view is an expression consisting of one or more of the following in valid combinations:

- Entity-action-view attribute-view (preceded by DESIRED, SOME, CURRENT, or THAT)
- Other-view attribute-view
- Function (character-, numeric-, date-, or time-related)
- Special attribute (USER ID, PRINTER TERMINAL ID, TERMINAL ID, TRANCODE, CURRENT DATE, CURRENT TIME)
- Numeric literal
- Character string
- Special keyword (SPACES, YEARS, MONTHS, DAYS, HOURS, MINUTES, SECONDS)
- Numeric operators: plus (+), minus (-), multiply (\*), divide (/), exponent (\*\*)
- Explicit indexing constructs (SUBSCRIPT, LAST, MAX)

An attribute condition must include an attribute of an entity action view on at least one side of the comparison operator. Comparisons must involve combinations of operands of the same domain (character, number, date, or time).

Each reference to an attribute of an entity action view is preceded by a view qualifier (DESIRED, SOME, CURRENT, or THAT). DESIRED refers to one of the views being read. SOME refers to any other entity action view. THAT refers to a view previously referenced by SOME. CURRENT can refer to the current occurrence of any entity action view.

The following examples show the usage of attribute conditions for a READ statement in Action Diagrams. The first example discovers whether the business knows about the CUSTOMER who is attempting to place an ORDER (in the process Take Order):

```
READ customer
WHERE DESIRED customer name IS EQUAL TO
requesting customer name
```

The following example READs all customers whose name begins with Q:

```
READ customer
WHERE SUBSTR(DESIRED customer name, 1,1) IS EQUAL TO "Q"
```

The last statement READs an explicitly indexed repeating group view and references a subscript for the group view which is incremented by one.

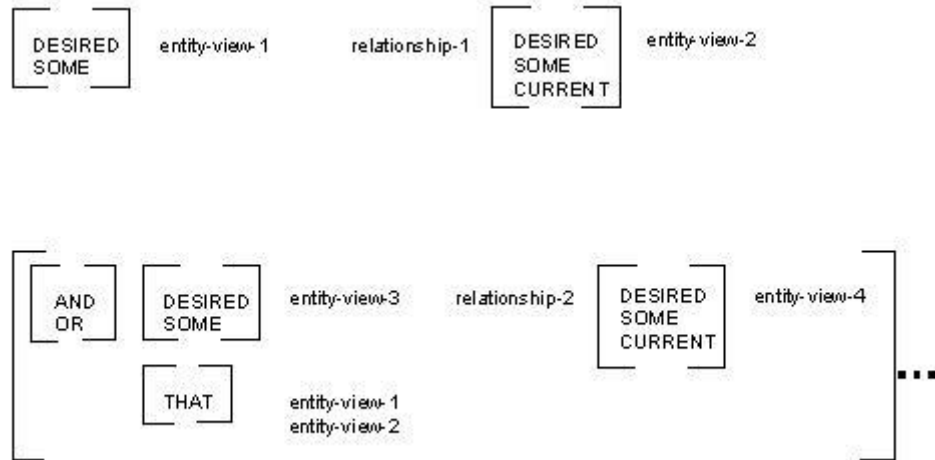
```
READ customer_order
WHERE input customer order IS EQUAL TO SUBSCRIPT OF
group_customer_order + 1
```

### Specify Relationship Conditions for a READ Action

The second part of specifying selection conditions for a READ (and READ EACH) action is to add any relationship conditions.



The following illustration shows a relationship-condition:



A relationship-condition must be a relationship between two entity action views.

Each reference to an entity action view is preceded by a view qualifier (DESIRED, SOME, CURRENT, or THAT). DESIRED refers to one of the views being read. SOME refers to any other entity action view. THAT refers to a view previously referenced by SOME. CURRENT refers to any entity action view previously populated by a READ or a CREATE action.

The following example shows how relationship conditions are used in Action Diagrams.

```

READ customer
order
WHERE DESIRED customer places DESIRED order
AND DESIRED order contains SOME order line
AND THAT order_line is for CURRENT product

```

This statement READs a CUSTOMER and ORDER pair satisfying the criterion of containing an ORDERLINE for the CURRENT PRODUCT.

**Note:** The Action Diagram tool does not offer THAT as a selection option, but the tool does insert THAT as an exception option. CA Gen inserts THAT in a WHERE clause if there is more than one qualifier beginning with SOME. CA Gen automatically translates the second and subsequent references to SOME entity view to read THAT entity view. This enhances the readability of complicated WHERE clauses.

**Restriction:** There is a restriction in the case where two entities, having a fully optional one-to-many relationship, are used in an entity action statement to retrieve all rows for the entity on the many sides of the relationship and the key of the other entity is tested for NULL. In this case, no other clauses or conditions are allowed in the entity action statement. For example, the use of IS NULL on a fully optional relationship to find records with NULL foreign keys is only supported for simple WHERE expressions. More complex WHERE expressions, containing an OR combined with the use of the IS NULL, in this particular scenario is not allowed.

## Specify System Conditions

The third part of specifying selection conditions for a READ statement is to specify any system conditions. You first specify the system value of the entity being read. CA Gen supplies four system values:

- User ID
- Printer terminal ID
- Terminal ID
- Transaction code

You then specify the relational operator and complete the expression by qualifying any entity action view attributes. You specify any character attributes, character strings, or spaces.

The following example of a READ statement contains a system condition for terminal ID:

```
READ library
WHERE TERMINAL ID IS EQUAL TO DESIRED library terminal_id
```

## Specify READ Exception Logic

Exception logic defines the actions to take if an exception to an entity action occurs. You can add the following exception conditions to a READ action:

- WHEN successful
- WHEN not found

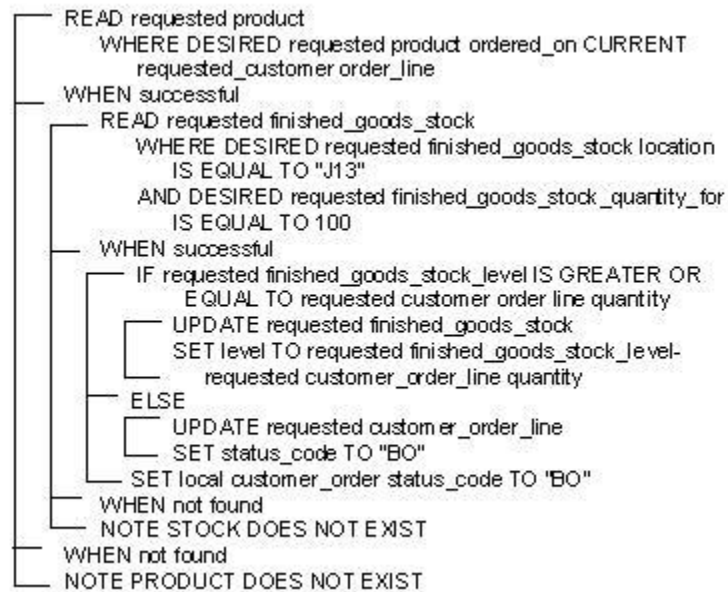
The WHEN successful exception condition allows you to specify actions to be taken after a successful READ. Not using the WHEN successful exception condition can cause maintainability problems and can decrease execution performance.

The WHEN not found exception condition allows you to specify actions to be taken if an entity occurrence that meets the selection criteria is not found. These conditions also apply to a combination of entity occurrences for each view in the READ list.

If no records are returned as a result of the READ, and there is no WHEN not found clause, an error occurs. The runtime routine TIRFAIL is called to report the error condition.

The following example illustrates how actions following exception conditions in a READ statement can be nested. This example contains multiple READs with multiple exception conditions.

These are Multiple READ Statements with Multiple Exception Conditions:



## Add UPDATE Entity Action

Adding an UPDATE entity action consists of the following activities:

- READ or CREATE entity
- Add UPDATE exception logic

The UPDATE entity action can change the attribute values of an entity occurrence. UPDATE can also change the relationships of an entity.

## READ or CREATE Entity

You must READ or CREATE an entity before you UPDATE it. It is also possible for the view being acted upon to be a persistent view READ or CREATED by an earlier action block.

When using an UPDATE statement, add appropriate READ and CREATE entity actions to the Action Diagram. Use SET to change attribute values.

The UPDATE action statement has the following format:

```
UPDATE entity-view-1
[SET attribute-view-1 TO expression ] ...
[REMOVE attribute-view-2]...
[ASSOCIATE WITH entity-view-2 WHICH relationship IT]...
[DISASSOCIATE FROM entity-view-3 WHICH relationship-2 IT]...
[TRANSFER FROM entity-view-4 WHICH relationship-3 IT
TO entity-view-4 WHICH relationship-4 IT]...
WHEN successful
action-statement-list
WHEN not unique
action-statement-list
```

**Downstream Effects:** The generated code eliminates the need to search the DBMS for a record to be UPDATED when the following conditions occur:

- The table being UPDATED has been READ successfully
- The table is not involved in another READ or CREATE
- The READ and the UPDATE use the same view
- The view is not used in any other READ statement except with WHERE conditions
- The WHERE CURRENT on the cursor name can be used because the cursor is already positioned on the row requiring the update.

## Add UPDATE Exception Logic

Exception logic defines the actions to take if an exception to an entity action occurs. You can add the following exception conditions to an UPDATE action:

- WHEN successful
- WHEN not unique
- WHEN permitted values violation

The WHEN not unique exception condition executes if the UPDATE action changes the value of an identifying attribute so that it is no longer unique. The WHEN successful exception condition executes if the UPDATE action is successful. The WHEN permitted values violation executes when the application retrieves data external to CA Gen (a file or external database) that may not be validated against permitted values. If you do not add logic for the WHEN permitted values violation condition, a runtime error is issued when the condition occurs.

The following statement is terminated with ESCAPE if the UPDATE of received product modifies the identifying attribute to a common value:

```

      UPDATE received product
      SET number delivered TO input product number_delivered
      ASSOCIATE WITH processing warehouse WHICH holds IT
      WHEN not unique
      ← ESCAPE
  
```

## Add DELETE Entity Action

Adding a DELETE entity action consists of the following activities:

- READ or CREATE entity
- Add DELETE action

The DELETE entity action removes an occurrence of an entity. After a DELETE, the business can no longer access the entity occurrence. Before performing the DELETE, the process logic must have CREATED or READ the entity, as shown in the following example.

The following example shows DELETE following a READ Statement:

```

      READ selected product
      WHERE DESIRED product number IS EQUAL TO input
        product number
      ASSOCIATE WITH processing warehouse WHICH holds IT
      WHEN successful
      DELETE selected product
      WHEN not found
      ← ESCAPE
  
```

**Note:** DELETE has no exception condition.

The DELETE action also deletes all entity occurrences that no longer participate in mandatory pairings with occurrences of this entity type. This effect is named cascade deletion. For example, ORDER and ORDER LINE have a mandatory relationship in the Data Model. It becomes necessary to delete the ORDER with an ORDER Number of 111.

The following logic addresses this situation:

```

READ order
WHERE DESIRED order number IS EQUAL TO 111
DELETE order
  
```

**Downstream Effects:** Because ORDER has a mandatory relationship to ORDER LINE, CA Gen generated code also deletes all ORDER LINES associated with the affected ORDER. If ORDER LINE had any other mandatory relationships, CA Gen would have deleted any entity paired with an ORDER LINE based on that relationship. The cascade delete continues until all pairings based on mandatory relationships have been deleted.

**More information:**

[DISASSOCIATE Entities](#) (see page 32)

[TRANSFER Relationships](#) (see page 32)

## Delete Followed by Insert

Unpredictable results can occur when subsequent READs follow Delete and Insert statements. To prevent unknown results, use a DISTINCT clause, or perform an UPDATE instead of a DELETE and an INSERT.

## READ Properties

Properties of a READ statement control cursor generation. To access the Read Properties dialog, highlight a READ statement. Select the Detail pull down menu and the Properties Action. For more information, see the *Toolset Help*.

You can specify how to control the generation of a database cursor resulting from a READ statement using the READ Properties dialog.

**Note:** For more information, see the *Toolset Help*.

## Add READ EACH Entity Action

The READ EACH entity action allows you to retrieve all entity occurrences that meet the selection criteria. The read is successful and iterates once for each time any of the views in the READ list receives a new occurrence.

**More information:**

[Add Repeating Actions](#) (see page 64)

## READ EACH Properties

Properties of a READ EACH statement determine whether the statement is generated with a DISTINCT clause. To access the Read Each Properties dialog, highlight a READ EACH statement. Select the Detail menu and the Properties Action.

**Note:** For more information, see the *Toolset Help*.

## Add Relationship Actions

A relationship action is an action used to manipulate pairings of stored entities. Relationship actions maintain the relationship between the entities in an entity action.

Adding relationship actions to the Action Diagram consists of the following activities:

- ASSOCIATE entities
- DISASSOCIATE entities
- TRANSFER relationships

Relationship actions appear as clauses in CREATE or UPDATE action statements. Specifically, ASSOCIATE can appear as a clause in either CREATE or UPDATE action statements. DISASSOCIATE and TRANSFER can also appear as clauses in UPDATE action statements. ASSOCIATE, DISASSOCIATE, and TRANSFER can also appear as standalone relationship actions.

**Note:** When ASSOCIATE, TRANSFER and DISASSOCIATE appear outside of CREATE and UPDATE statements, all appropriate READS must be performed to populate the entity action views involved.

## ASSOCIATE Entities

After you CREATE an entity, you build its identifying relationship memberships using ASSOCIATE. The entity being associated must be one the Action Diagram recognizes through a CREATE or READ. ASSOCIATE establishes a pairing along a relationship between two entities. You ASSOCIATE entities when the relationship is optional, and you must ASSOCIATE them (before the end of the elementary process) when the relationship is mandatory.

**Important!** Do not attempt to ASSOCIATE entities that have a mutually exclusive relationship in the Data Model. This can cause the application to end abnormally because code is generated to enforce the mutually exclusive condition.

ASSOCIATE can appear as a clause in either CREATE or UPDATE action statements. The following CREATE statement shows an example of ASSOCIATE action:

```
CREATE received product
SET code TO "RECD"
ASSOCIATE WITH processing warehouse WHICH holds IT
```

You can also create a standalone ASSOCIATE action. It operates the same as the relationship clauses in the CREATE and UPDATE, with one exception: in relationship clauses, the subject of the clause is always the subject of the entity action. In a standalone relationship action, you explicitly specify the subject of the action.

The following example shows a standalone ASSOCIATE action:

```
ASSOCIATE existing product  
WITH processing warehouse WHICH holds IT
```

## DISASSOCIATE Entities

You can DISASSOCIATE entities whose relationships have been designated as optional. You cannot DISASSOCIATE an identifying relationship.

The DISASSOCIATE action breaks a specific pairing along a specific relationship. In the example below, the relationship between PRODUCT and WAREHOUSE in the Data Model still exists but the pairing does not.

DISASSOCIATE can appear as a clause in an UPDATE action statement. The following UPDATE statement shows an example of a DISASSOCIATE action:

```
UPDATE received product  
SET number_delivered TO input product number_delivered  
DISASSOCIATE FROM warehouse WHICH can hold IT
```

DISASSOCIATE can also appear as a standalone relationship action. It operates the same as the relationship clauses in the CREATE and UPDATE, with one exception: in relationship clauses, the subject of the clause is always the subject of the entity action. In a standalone relationship action, you explicitly specify the subject of the action.

The following statement shows an example of a standalone DISASSOCIATE action.

```
DISASSOCIATE received product  
FROM order_line WHICH contains IT
```

In the previous example, received product is deleted if its relationship to order\_line is mandatory and has a cardinality of one. In addition, any product components dependent on that occurrence of PRODUCT are deleted.

## TRANSFER Relationships

You can TRANSFER relationships defined as transferable in the Data Model. The TRANSFER action exchanges the relationship between specific entity occurrences. The effect of TRANSFER is to change the pairing from one entity to another. This action does not affect the relationship between the entities in the Data Model.

Identifying relationships may not be the subject of a TRANSFER clause. Also, the relationship membership cannot have a many-to-many cardinality.

You must READ or CREATE the entity before you TRANSFER the relationship. You can TRANSFER as part of an UPDATE or as a standalone action. If you TRANSFER a relationship between entities where the relationship is mutually exclusive, the application terminates.



To illustrate TRANSFER relationships, consider multiple warehouses, each of which holds products. Products can be moved between warehouses. In the following statement, PRODUCT has been moved from PROCESSING WAREHOUSE to NEW WAREHOUSE:

```
TRANSFER received product
FROM processing warehouse WHICH holds it
TO new warehouse WHICH holds it
```

The relationship between WAREHOUSE and PRODUCT is represented in the Data Model as PRODUCT always is held in one WAREHOUSE and WAREHOUSE sometimes holds one or more PRODUCT. The relationship is defined as transferable and is not changed in the Data Model.

## Add Assignment Actions

An assignment action assigns values to attribute views. Assignment actions act on the attributes of an entity involved in an entity action. These actions include SET, MOVE, COMMAND IS, EXIT STATE IS, and PRINTER TERMINAL IS.

Adding assignment actions, part of defining logic for elementary processes, consists of the following activities:

- Add attribute actions
- Add MOVE actions
- Add special attribute actions
- Add explicit indexing expressions
- Add NEXTTRAN system attributes

## Add Attribute Actions

Adding attribute actions consists of the following subordinate activities:

- SET text attributes
- SET numeric attributes
- SET date attributes
- SET time attributes
- SET export or local view attributes

When you CREATE an entity, you assign values to its identifier attributes. When you CREATE or UPDATE an entity, you can SET any attribute value.

When you work with attribute actions, you use the SET action to assign a value to an individual attribute view. SET either establishes an initial value for an attribute or changes an existing value. You can SET an attribute by assigning a specific value using the TO option, or by calculating a value USING an algorithm.

When using the TO option, CA Gen presents expressions that are consistent with the primitive domain of the attribute you selected.

The SET action is used in CREATE or UPDATE action blocks and also can be used as a standalone action. The SE action is similar to SET clause.

**More information:**

[Add CREATE Entity Action](#) (see page 17)

[Add UPDATE Entity Action](#) (see page 27)

## Set Text Attributes

Setting text attributes consists of the following activities:

- Select text attribute in entity action view
- Set text attribute to character expression (character function, character string, character view, special attribute, and spaces)
- Set text attribute using action block or algorithm

For information about character functions, see the *Toolset Help*.

The SET statement in the following example sets a text attribute to a character string by assigning a literal value to code:

```
CREATE received product
SET code TO "GA"
```

A character view is a component of an entity view that sees an attribute that is defined as text. The SET statement in the following example sets a text attribute to a character view by finding the value of code and copying it:

```
CREATE received product
SET code TO input product code
```

Special attributes include terminal ID, user ID, printer ID, and trancode. The SET statement in the following example specifies a value for a printer terminal:

```
SET id TO PRINTER_TERMINAL_ID
```

The SET statement in the following example sets a text attribute to spaces:

```
SET status TO SPACES
```

The SET statement in the following example uses an algorithm to set an attribute value:

```
CREATE new department
SET code USING find_department
WHICH IMPORTS: Entity View division
```

SET can also be used as a standalone action to establish a value for an attribute from an export or local view. The following statement shows an example:

```
SET export code T0 input product code
```

## Set Numeric Attributes

Setting numeric attributes consists of the following activities:

- Select numeric attribute in entity action view
- Set numeric attribute to numeric expression (explicit indexing expression, specific expression, number, numeric function, and numeric view)
- Set numeric attribute using action block or algorithm

**Note:** For information on numeric functions, see the *Toolset Help*.

You can set numeric attributes for entity views used in repeating group views to explicit indexing expressions. You must have previously set the group view to explicitly indexed in View Maintenance when you set its cardinality. A numeric attribute of an entity view can be set to:

- Subscript of repeating group view
- Last populated item of repeating group view
- Absolute maximum of repeating group view

The SET statement in the following example sets a numeric attribute to an expression used to reference an explicitly indexed repeating group view:

```
SET work index T0 SUBSCRIPT OF screen_group + 1
```

The SET statement in the following example shows an expression for a numeric attribute built directly in the Action Diagram syntax:

```
SET product sales_tax T0 (price * .06)
```

The SET statement in the following example sets a numeric attribute to a number:

```
SET new number T0 12
```

A numeric view is a component of an entity view. A numeric view sees an attribute that is defined as numeric. The SET statement in the following example sets a numeric attribute to a numeric view by finding the value of customer number and copying it:

```
CREATE customer
SET number TO input customer number
```

## Set Date Attributes

Setting date attributes consists of the following activities:

- Select attribute in entity action view
- Set date attribute to date expression (current date, date function, and date view)
- Set date duration expression
- Set date attribute using date action block or algorithm

**Note:** For information about date functions, see the *Toolset Help*.

A SET action can be used to establish the value of an attribute from an export or local view.

The SET statement in the following example sets a date attribute to the current date:

```
SET actual_date_started TO current date
```

A date view is a component of an entity view. A date view sees an attribute that is defined as a date. The SET statement in the following example sets a date attribute to a date view:

```
SET actual_date_started TO start date
```

You can build date duration expressions in Action Diagrams. These expressions let you add a number of years, months, and/or days to a date value. The expressions include dates and labeled durations. The result is another date value. Date durations are available when you select a date value (either date view or current date).

The following statement illustrates the format for date duration expressions:

```
SET customer_contact_date TO import_customer_contact_date
_2_YEARS_3_MONTHS 12 DAYS
```

## Set Time Attributes

Setting time attributes consists of the following activities:

- Select time attribute in an entity action view
- Set time attribute to time expression (current time, time function, and time view)

- Set time duration expression
- Set time attribute using time algorithm

**Note:** For information about time functions, see the *Toolset Help*.

A SET action can be used to establish the value of an attribute from an export or local view.

The SET statement in the following example sets a time attribute to the current time:

```
SET actual_time_started TO CURRENT_TIME
```

A time view is a component of an entity view that sees an attribute that is defined as time. The SET statement in the following example sets a time attribute to a time view:

```
SET actual_time_started TO project_actual_time_started
```

You can build time duration expressions in the Action Diagram. These expressions let you add a number of hours, minutes, and seconds to a time value. The expressions include times and labeled durations. The result is another time value. Time durations are available when you select a time value (either time view or current time).

The following statement illustrates the format for time duration expressions:

```
SET customer_contact_time TO import_customer_contact_time  
_2_HOURS_13_MONTHS + 1 SECOND
```

## Set Export or Local View Attributes

When you set export or local view attributes, you create a standalone SET action.

## Add a MOVE Action

Adding a MOVE action to the Action Diagram is part of creating assignment actions. This action moves information into the export view so you can output information from a process. In the following example, the MOVE statement takes the information from new customer and places it into the export view preferred customer list:

```
MOVE new customer to preferred customer list
```

The MOVE statement moves the entire view. You do not have to add a MOVE action for each attribute value. Any extra attributes in the destination view (preferred customer list in the example above) are not changed.

Transient views can be modified by MOVE and SET actions. Persistent views cannot be the target of MOVE or local SET verbs.

## Add Special Attribute Actions

Adding special attribute actions to the Action Diagram, also part of creating assignment actions, consists of the following activities:

- Add EXIT STATE IS action
- Add COMMAND IS action
- Add PRINTER TERMINAL IS action

Special attributes are implementation-specific attributes. You can test these attributes in condition statements (such as IF).

## Add EXIT STATE IS Action

An exit state is a trigger that transfers control from one Procedure Step to another. A dialog flow takes place based on the existence of one or more exit states.

**Note:** You can add two types of exit states with the Action Diagram in Analysis and Design: global exit states and exit states created for a specific business system.

Global exit states may be defined prior to the existence of a business system and thus are not associated with a business system. They accommodate the concept of common action blocks. Because a common action block may be USED in multiple business systems within the same model, the exit states referenced and set within that action block must be available to all those business systems.

Both types of exit states can be used by any Action Diagram or action block within the same model, regardless of business system.

**Note:** For more information about managing exit states and their messages, see the *Design Guide*.

Exit states are set within procedures based on the following:

- Interpretation of a command, which may be set by a PF key
- Derived by processing logic within the Procedure Step, which may be based on data input

You can modify an exit state in the Action Diagram with the EXIT STATE IS action. This action sets the value of the special attribute, exit state. In the EXIT STATE IS statement, you can compare the exit state to possible exit state values. Since there is only one occurrence of the exit state special attribute, the execution of an EXIT STATE IS action destroys its previous contents.

The format of the EXIT STATE IS action statement is:

EXIT STATE IS exit-state-value

The following statement sets the exit state value to order\_cancelled:

```
EXIT STATE IS order_cancelled
```

The following statement uses the EXIT STATE IS statement in an IF statement to set screen attributes:

```
IF EXIT STATE IS EQUAL TO  
customer_not_found  
MAKE ERROR
```

## Add COMMAND IS Action

A command is a request for action made to a Procedure Step by another Procedure Step or a user. A command tells the Procedure Step what logic to use.

A command allows you to direct the execution of a procedure. A procedure can implement more than one process. You specify the command that instructs the procedure which process to execute at any given time. You can use a command to execute an action block in the current Procedure Step or another Procedure Step.

In the Procedure Action Diagram, you create a COMMAND IS statement to set the value of the special attribute, command. Because there is only one occurrence of the command special attribute, the execution of a COMMAND IS action overlays its previous contents.

The COMMAND IS action takes the following form:

```
COMMAND IS command_value:
```

The following statement sets the command value to blanks:

```
COMMAND IS SPACES
```

The following statement sets the command value to “cancel”

```
COMMAND IS cancel
```

**Note:** You can add commands only when you are working with system defaults, Dialog Design, and the Action Diagram. You can add the COMMAND IS action only in the Procedure Action Diagram and in Design action blocks. Although any action blocks containing COMMAND IS actions that you create in Design can be accessed with the Action Diagram in Analysis, you cannot add or manipulate commands in Analysis. To manage commands and their synonyms, see the *Design Guide*.

## Add PRINTER TERMINAL IS Action

The PRINTER TERMINAL IS action statement sets the value of the special attribute, printer terminal ID. If a value for this special attribute is present when a Procedure Step completes execution, the Procedure Step prints a copy of its associated screen on the identified terminal. This statement has meaning only in online procedures.

The format of the PRINTER TERMINAL IS action is:

```
PRINTER TERMINAL IS printer-terminal-value
```

The following statement uses a character view for printer output:

```
PRINTER TERMINAL IS output TERMINAL_ID
```

The following statement uses spaces for printer output:

```
PRINTER TERMINAL IS SPACES
```

The following statement uses a literal printer ID value for printer output:

```
PRINTER TERMINAL IS PRINT1
```

## Add Expressions for Explicit Indexing

Adding expressions for explicit indexing consists of the following activities:

- SET subscript of explicitly indexed repeating group view
- SET last variable of explicitly indexed repeating group view

Explicit indexing allows you to incorporate complex array processing into the Action Diagram and resulting generated code. You can specify the Action Diagram expressions for explicit indexing to refer to specific entries in a repeating group view (or table). Explicitly indexed repeating group views have user-accessible indexes that are used as any other numeric attribute view.

The explicit indexing expressions are:

- Subscript Index of a data array (all views in an explicitly indexed repeating group view) that indicates the specific item (view) in the referenced group view. The value is the specific item of interest. You modify directly it in the Action Diagram.
- Last Index variable that specifies the last item in the array containing data. The value is changed automatically when a row is modified that has a higher index than the value in LAST.
- Max Index variable that specifies the highest array location, whether it contains data. Max refers to the actual size of the repeating group view which is the maximum cardinality you set when you defined properties for the group view. The value is constant and cannot be modified during execution.



For example, consider a group view (or table) that contains 10 occurrences of entity views (or items) and you want to reference the second occurrence. You would specify SUBSCRIPT set to 2 in the SET statement to have the subscript point directly to the second occurrence in the repeating group view. In another statement, you want to reference the last view in the same repeating group view where the first five items contain data. Using LAST in the SET statement points to the fifth view in the group view. Finally, in another statement, you want to reference the last view in the group view that now has six items filled with data. Using the MAX index variable points to the tenth view (the maximum for the repeating group view) even though only six views are filled.

You observe the following rules concerning explicit indexing:

1. All references to entity views and their attribute views defined within the explicitly indexed repeating group view are implied to mean the occurrence of that view indicated by the current value of SUBSCRIPT for the group view.
2. You can SET SUBSCRIPTs OF views and LAST OF views to numeric expressions; you can also SET numeric attribute views to SUBSCRIPT OF views.
3. MAX is not valid in SET statements because the value is constant.
4. The value of the first SUBSCRIPT of a repeating group is 1 (not 0 as in some systems). The value of LAST is 0 or positive. The value of MAX is always the maximum cardinality of the view.
5. The SUBSCRIPT value must be set to a value greater than 0 and less than or equal to MAX.

The SET statements in the following example set a SUBSCRIPT OF a repeating group view defined as explicitly indexed. This example illustrates the position of the SET statements containing SUBSCRIPT expressions. It also illustrates how the subscript increments by a value of 1.

**Note:** The *export\_list* is explicitly indexed and *local\_list* is implicitly indexed.

```
READ EACH customer
WHERE DESIRED customer places SOME order
TARGETING local_list FROM THE BEGINNING UNTIL FULL
SET SUBSCRIPT OF export_list TO 1
FOR EACH local_list
MOVE local customer TO export customer
SET SUBSCRIPT OF export_list TO SUBSCRIPT OF export_list + 1
```

## Add NEXTTRAN System Attribute

CA Gen provides the NEXTTRAN (next transaction) special attribute as the means to invoke non-CA Gen transactions from a CA Gen-generated transaction. The NEXTTRAN attribute is specified within a Dialog Flow Diagram.

**Note:** For more information about the Dialog Flow Diagram and NEXTTRAN, see *Design Guide*.

You can use the NEXTTRAN attribute to implement the following types of flows:

- Transfers to non-CA Gen-generated transactions
- Transfers or links to CA Gen-generated transactions in a different business system

When transferring to non-CA Gen-generated transactions, the NEXTTRAN attribute can specify the transaction code to be invoked by the Dialog Manager when the current transaction terminates. You can place the NEXTTRAN attribute on a screen and can have the user enter the desired transaction code in the field. Alternately, you can set the NEXTTRAN attribute using the SET command. In either case, the transaction code must conform to the requirements of the teleprocessing monitor and the receiving transaction.

The NEXTTRAN field must be formatted correctly for the receiving transaction. The transaction code and any unformatted input data must be valid to the receiving transaction.

The use of external action blocks is another method by which you can access existing subroutines and databases.

**More information:**

[Define External Action Blocks](#) (see page 80)

## Add NEXTLOCATION System Attribute

CA Gen provides the NEXTLOCATION special attribute to specify the location of a server transaction in a distributed processing client/server environment. This system attribute can be used to direct server transactions and client requests when transactions reside on multiple application servers.

You can SET NEXTLOCATION to any character string up to a maximum length of 1,000 characters. The character string is a logical address that vary by site.

During processing, the Client Manager on the client uses an internal mapping table or a user-defined directory service to map the value of NEXTLOCATION to the appropriate application server.

**Note:** NEXTLOCATION has only one value at a time. Initially, the value is set to blanks. After a value is set, the value stays in effect during the execution of the load module until a different value is set. Ensure that you SET NEXTLOCATION to the correct value if you want a different server transaction than the one currently set.

**Note:** For more information, see the *Design Guide*.

## Add SUMMARIZE Entity Action

Adding a SUMMARIZE entity action is part of creating entity actions in the Action Diagram. This statement lets you summarize all selected occurrences of a set of entities consisting of one or more related entity types.

The SUMMARIZE action statement must be in the following format:

```
SUMMARIZE entity-view-list  
PLACING aggregate function INTO attribute-view
```

### **entity-view-list**

Contains the same views eligible for READ or READ EACH statements

### **PLACING clause**

References an aggregate function and designates a modifiable attribute view to receive the results. The attribute-view is transient and modifiable.

The SUMMARIZE statement supports the use of all ANSI standard SQL aggregate functions such as COUNT, MAX, MIN, AVERAGE, and SUM to be used to derive values from a specified collection of entities. The SUMMARIZE statement is used to retrieve a single composite value for each aggregate function specified. It is similar to READ except that one or more PLACING clauses are required, in which modifiable attribute views of the appropriate domain are designated to receive each aggregate value.

The SUMMARIZE statement reviews all selected occurrences of a set of entities consisting of one entity type or multiple related entity types, and retrieves aggregate values from the database using one or more of the following aggregate functions:

- Average
- Average Distinct
- Count Distinct
- Count Occurrences
- Maximum
- Minimum
- Sum
- Sum Distinct

## Examples

For use of the SUMMARIZE statement, the following examples are organized by type of entity view and qualification.

## Add Single Entity Views

You can add single entity views with or without a WHERE clause as discussed in the following examples:

### Single Entity View without a WHERE Clause

To determine the total number of employees listed in the database, use a statement similar to the following example:

```
SUMMARIZE employee
PLACING count(OCCURRENCES) INTO local ief_supplied count
```

### Single Entity View with a WHERE Clause

To find the total salaries of all exempt employees, use a statement similar to the following example:

```
SUMMARIZE employee
PLACING sum(employee salary) INTO local employee salary
WHERE DESIRED employee classification = 'EX'
```

## Add Multiple Entity Views

You can add multiple entity views as discussed in the following example:

### Multiple Entity Views with a Simple Relationship

Using an aggregate function with multiple views provides a result table or an intersection result table depending on the relationship specified in the WHERE clause between those views.

For example, to find the total number of classes attended by all the students, use a statement similar to the following example:

```
SUMMARIZE student
class
PLACING count(OCCURRENCES) INTO local ief_supplied count
WHERE DESIRED student
attends DESIRED class
```

**Note:** When multiple entity views are included in the read list, each aggregate function in a PLACING clause reference the intersection table that results by joining their corresponding tables. Therefore, the number of occurrences in the result equal the product of the number of entities of each type that are selected. The joining of the tables occurs because of the relationships defined between the entities.

**Related Entities Not in the Read List**

To find the number of students that actually attend at least one class, use a statement similar to the following example:

```
SUMMARIZE student
PLACING count(OCCURRENCES)
INTO local_ief_supplied count
WHERE DESIRED student attends SOME class
```

**Note:** The result table excludes entity views not in the entity view list. Relationships between DESIRED and SOME entity views do not create an intersection table.

## Use Multiple Functions

You can use multiple functions in a single SUMMARIZE statement as discussed in the following examples:

**Same Entity View**

To find the lowest and highest employee salaries, use a statement similar to the following example:

```
SUMMARIZE employee
PLACING minimum (employee salary)
INTO local_min employee salary
PLACING maximum(employee salary)
INTO local_max employee salary
```

**Multiple Entity Views**

To find the average number of semester hours and grade-point average for all seniors, use a statement similar to the following example:

```
SUMMARIZE student
class
PLACING average (class semester_hours)
INTO local_class semester_hours
PLACING average(student grade_point_avg)
INTO local_student grade_point_avg
WHERE DESIRED student attends DESIRED class
AND DESIRED student grade_level = 12
```

## Use Expressions as Functions

Aggregate functions other than Count Occurrences, Count Distinct, Sum Distinct, and Average Distinct use expressions as arguments provided that they meet the following restrictions:

- At least one attribute of an entity action view in the read list is included.
- All entity action (non-transient) views referenced in the expression must either be in the read list or be designated as CURRENT.

- No non-aggregate functions (such as NUMTEXT) are included.
- The expression is of the proper domain.

Aggregate functions not be used as part of a larger expression.

**Note:** CURRENT views allowed in an aggregate function expression must be populated before the SUMMARIZE statement executes.

#### Single Entity View

To find the latest start time for all classes expressed in daylight standard time, use a statement similar to the following example:

```
SUMMARIZE class
PLACING maximum(class start_time
+ local_daylight_time adjustment_hours)
INTO local_avg class start_time
```

#### Multiple Entity Views

Use a statement similar to the following example to find the total billable amount (quantity times price plus sales tax):

```
SUMMARIZE order_line
product
PLACING sum((order_line qty_ordered * product unit_price)
* (1 + CURRENT purchase_order sales_tax_rate))
INTO local_purchase_order amt_billed
WHERE DESIRED order_line belongs_to CURRENT purchase_order
AND DESIRED product is_ordered_in DESIRED order_line
```

**Note:** Both the entity views in the read list (order\_line and product) are used as the subject of the aggregate function sum.

### Add SUMMARIZE EACH Entity Action

You can use the SUMMARIZE EACH statement with the SQL aggregate functions COUNT, MAX, MIN, AVERAGE, and SUM for the following purposes:

- To subdivide all selected occurrences of a set of entities into groups based on common attribute values
- To retrieve aggregate values for each group

You can also use the SUMMARIZE EACH statement to retrieve the common attribute values themselves, without selecting an aggregate function. This feature can eliminate the need for a DISTINCT clause on a READ EACH statement, because that solution does not yield the desired results.

**Note:** For more information about using a DISTINCT clause in READ EACH statement, see the *Toolset Help*.

The SUMMARIZE EACH statement is similar to the SUMMARIZE statement in that it requires a PLACING clause and allows a WHERE clause, but it also includes a WITH THE SAME clause to designate the common attributes.

The SUMMARIZE EACH statement also requires each of the common attribute views to be placed with a PLACING clause, and allows TARGETING and SORTED BY clauses.

Within SUMMARIZE EACH statements, aggregate functions that apply to groups are preceded by the word GROUP, to distinguish them from aggregates that apply to all the selected entities. All aggregate functions in the PLACING clause of a SUMMARIZE EACH GROUP statement will be GROUP functions.

The SUMMARIZE EACH statement supports the use of all ANSI standard SQL aggregate functions. This statement lets you subdivide all selected occurrences of a set of entities into groups based on common attribute values, and retrieve aggregate values for each group. The format of the SUMMARIZE EACH statement is:

```
SUMMARIZE EACH GROUP OF entity-action-view
WITH THE SAME entity-action-view-attribute(s)
TARGETING group-view FROM THE BEGINNING UNTIL FULL
PLACING entity-action-view-attribute INTO export entity-action-view-attribute
PLACING GROUP aggregate function INTO export entity-action-view number
SORTED BY {ASCENDING/DESCENDING} placing-clause-source-value
WHERE selection-conditions
action-statement-list
```

**entity-action-view**

Contains the same views eligible for READ or READ EACH. The views that appear in the entity-view-list not be populated or modified.

**WITH THE SAME clause**

Identifies your groups. All common attributes designated in the WITH THE SAME clause appear in the PLACING clauses before any aggregate functions.

**TARGETING clause**

(Optional) Specifies the implicitly indexed repeating groups to be populate.

**PLACING clause**

References a WITH THE SAME attribute view and designates a modifiable attribute view to receive the results.

**PLACING GROUP clause**

(Optional) References an aggregate function and designates a modifiable attribute view to receive the results. The keyword, GROUP, preceding the aggregate function indicates that a new value for the function, and thus a new result table, is computed for each iteration.

**SORTED BY clause**

(Optional) Indicates whether the sort is ASCENDING or DESCENDING. It must reference either a PLACING (WITH THE SAME) attribute view or an aggregate function expression that occurs in a PLACING GROUP clause. The attribute-views are transient and modifiable.

**WHERE clause**

(Optional) Indicates the entities to be aggregated.

**action-statement-list**

Indicates a block of actions to execute for each group occurrence.

**Note:** The TARGETING, PLACING GROUP, SORTED BY, and WHERE clauses are all optional. If there is no SORTED BY clause, the groups are sorted in ascending sequence by the attributes in the WITH THE SAME clause.

The SUMMARIZE EACH statement supports the following SQL aggregate functions:

- Average
- Average Distinct
- Count Distinct
- Count Occurrences
- Maximum
- Minimum
- Sum
- Sum Distinct

**Note:** For more information about these aggregate functions, see the *Toolset Help*.

The SUMMARIZE EACH statement stops iterating when either of the following situations occurs:

- No more occurrences exist that match the selection criteria.
- A database exception occurs, regardless of whether any database exception statements exist for the SUMMARIZE EACH statement.
- If database exception statements are added to a SUMMARIZE EACH statement, they appear after any other statements that are enclosed by, and subordinate to, the SUMMARIZE EACH statement. A new "(for each successful iteration)" line also displays immediately before any such subordinate statements. Contract the line "(for each successful iteration)" to display database exceptions next to a SUMMARIZE EACH statement.



**Note:** For the WHERE clause in a SUMMARIZE EACH statement, views that are nullable be tested for NULL or NOT NULL. You can test these views for equivalence by using the relational operators IS EQUIVALENT TO or IS NOT EQUIVALENT TO. For more information, see the *Toolset Help*.

## Group Aggregation Without Selection Criteria

The aggregate function COUNT, when used with OCCURRENCES, is used to count the number of entities in each group. The keyword, GROUP, preceding this function indicates that a new value for the function is computed for each iteration.

In the following example, the SUMMARIZE EACH statement finds the number of employees for each job grade:

```
SUMMARIZE EACH GROUP OF employee
WITH THE SAME employee job_grade
PLACING employee_job_grade INTO local employee job_grade
PLACING GROUP count(OCCURRENCES) INTO local ief_supplied count
```

**Note:** The SUMMARIZE EACH statement requires all common attributes designated in the WITH THE SAME clause to appear in PLACING clauses before the aggregate functions. For this example, the counts and corresponding Job Grades for each subgroup is stored in the designated local views, and retrieved in ascending Job Grade sequence because there is no SORTED BY clause.

## Group Aggregation TARGETING Repeating Group Views

The SUMMARIZE EACH statement is iterative and it provides a TARGETING clause that lets a new occurrence of repeating group views be populated each time the statement executes. However, unlike the READ EACH statement, SUMMARIZE EACH has the ability to populate a repeating group view's occurrences directly, by using its attributes in the PLACING clause.

In the following example, the PLACING . . . INTO attributes belong to a repeating group.

```
SUMMARIZE EACH GROUP OF employee
WITH THE SAME employee job_grade
TARGETING repeating_group_local FROM THE BEGINNING UNTIL FULL
PLACING employee job_grade INTO local employee job_grade
PLACING GROUP count(OCCURRENCES) INTO local ief_supplied count
```

In this example, the designated local views belong to the implicitly indexed repeating group view, repeating\_group\_local.

**Note:** Attributes of repeating group views referenced in the TARGETING clause may not be used in the WHERE clause. This is because the targeted occurrence of the repeating group is not yet available when the WHERE clause is executed. This restriction also applies to READ EACH.

The following example shows attribute view definitions associated with implicit:

```
LOCALS:
Group View repeating_group_local (4, implicit)
Work View local ief_supplied
count
Work View local employee
job_grade
ENTITY ACTIONS:
Entity View employee
number
job_grade
```

## Group Aggregation Using Explicitly Indexed Repeating Group Views

You can populate an explicitly indexed repeating group view by using a PLACING clause, if each iteration of the SUMMARIZE EACH statement is preceded by a SET SUBSCRIPT for that view:

```
SET SUBSCRIPT OF repeating_group_local to 1
SUMMARIZE EACH GROUP OF employee
WITH THE SAME employee_job_grade
PLACING employee_job_grade INTO local employee_job_grade
PLACING GROUP count(OCCURRENCES) INTO local employee_count
SET SUBSCRIPT OF repeating_group_local to SUBSCRIPT OF repeating_group_local + 1
SET LAST OF repeating_group_local to SUBSCRIPT OF repeating_group_local - 1
```

**Note:** The SET LAST statement is needed because the last SET SUBSCRIPT statement executed will follow the last successful SUMMARIZE EACH iteration, and thus will index an unused occurrence. This is important when assigning cardinalities to explicitly indexed views populated by the SUMMARIZE EACH statement.

The following example shows attribute view definitions associated with explicit:

```
LOCALS:
Group View repeating_group_local (4, explicit)
Work View local ief_supplied
count
Work View local employee
job_grade
ENTITY ACTIONS:
Entity View employee
number
job_grade
```

## Group Aggregation With Selection and Sort Criteria

You can change the previous example using implicit indexing to select only job grades 10 and above and to rank them by the number of employees as shown in the following example:

```
SUMMARIZE EACH GROUP OF employee
WITH THE SAME employee job_grade
TARGETING repeating_group_local FROM THE BEGINNING UNTIL FULL
PLACING employee job_grade INTO local employee job_grade
PLACING GROUP count(OCCURRENCES) INTO local ief_supplied count
SORTED BY DESCENDING GROUP count(OCCURRENCES)
SORTED BY ASCENDING employee job_grade
WHERE DESIRED employee job_grade IS GREATER OR EQUAL TO 10
```

In this example, GROUP count(OCCURRENCES) is the primary sort key because it occurs first. We are using employee job grade as a secondary sort key in case some job grades have the same number of employees.

## Examples

The following SUMMARIZE EACH examples are broken down by type of entity view and qualification.

### Subgroup Aggregation Without Qualification

Use the following statement to determine the total number of employees for each job grade:

```
SUMMARIZE EACH GROUP OF employee
WITH THE SAME employee job grade
PLACING employee job grade INTO local
employee job grade
PLACING GROUP count(OCCURRENCES)
INTO local ief_supplied count
```

### Subgroup Aggregation With Entity Qualifiers

Use the following statement to determine the total salary of exempt and non-exempt employees in department 3:

```
SUMMARIZE EACH GROUP OF employee
WITH THE SAME employee classification
PLACING employee classification
INTO local employee classification
PLACING GROUP sum(employee salary)
INTO local employee salary
WHERE DESIRED employee belongs to SOME department
AND THAT department id = 3
```

## Subgroup Aggregation With Multiple Entities

Multiple entities in the read list of a SUMMARIZE EACH GROUP statement should have a relationship between them.

Use the following statement to find the total salary by departments:

```
SUMMARIZE EACH GROUP OF employee
department
WITH THE SAME department id
PLACING department id INTO out department id
PLACING GROUP sum(employee salary) INTO out employee salary
WHERE DESIRED employee belongs to DESIRED department
```

## Subgroup Aggregation With Multiple Common Attributes

A subgroup can be based on more than one common attribute. Each common attribute used must appear in both the WITH THE SAME clause and the PLACING clause.

### Multiple Common Attributes of the Same Entity

Use the following statement to find the number of classes by subject that start at a given time:

```
SUMMARIZE EACH GROUP OF class
WITH THE SAME class_subject
class_start_time
PLACING class_subject INTO local class subject
PLACING class_start_time INTO local class start time
PLACING GROUP count(OCCURRENCES)
INTO local ief_supplied count
```

### Multiple Common Attributes and Entities

Use the following statement to find the total salary of exempt and non-exempt employees by department:

```
SUMMARIZE EACH GROUP OF employee
department
WITH THE SAME department id
employee classification
PLACING department id INTO out department id
PLACING employee classification INTO out employee classification
PLACING GROUP sum(employee salary) INTO out employee salary
WHERE DESIRED employee belongs_to DESIRED department
```

**Note:** Department is not used by any aggregate function, but needs to be in the read list because it provides a common subgroup attribute.

## Use Expressions as Aggregate Function Arguments

The rules for using expressions as aggregate function arguments for a SUMMARIZE EACH statement are the same as those for a SUMMARIZE statement. For more information, see Use Expressions as Functions in the section Add SUMMARIZE Entity Action.

For example, to retrieve the total billing amount (the sum of the quantities times the price of the ordered products) for each product, you can use the following statement:

```
SUMMARIZE EACH GROUP OF order_line
product
WITH THE SAME product id
PLACING product id INTO local product id
PLACING GROUP sum(order_line qty ordered * product unit price)
INTO local ief_supplied total_currency
WHERE DESIRED product is_ordered_in DESIRED order_line
```

## DATABASE EXCEPTION Statement

The DATABASE EXCEPTION statement specifies what action to take if a database-related problem occurs, as opposed to logical exceptions.

The following list details the DATABASE EXCEPTION statements:

- WHEN successful—use this statement when valid conditions exist for adding it. It can also be considered a logical exception.
- WHEN DATABASE DEADLOCK or TIMEOUT—use this statement when trying to update concurrently, or otherwise unable to access the data in a required amount of time.
- WHEN DATABASE ERROR—use this statement for any exception other than logical exceptions.

You must add the DATABASE EXCEPTION statements to the entity action statements. Do not use these statements unless you need to specify your own logic for handling these errors. CA Gen generates code to handle these conditions for you. It rolls back any database updates that occur up to that point; for timeouts and deadlocks, CA Gen may attempt to retry the transaction automatically. Otherwise, a fatal error occurs causing the application to terminate.

Add the DATABASE EXCEPTION statements if you need changes to CA Gen's default error-handling procedures.

**Important!** You are responsible for database integrity after you include database exception conditions. This includes any rollback that must be done.

Rollbacks will not occur unless you use one of the following clauses:

- ABORT TRANSACTION verb
- RETRY TRANSACTION verb
- EXIT STATE with rollback
- EXIT STATE with abort
- External action block that issues a rollback

Otherwise, any successful database updates that occur before or after the database exception are committed.

The CA Gen code generators provide default actions for the two database exception statements—WHEN DATABASE DEADLOCK or TIMEOUT and WHEN DATABASE ERROR. When a database deadlock or timeout exception is added, the generators automatically add the RETRY TRANSACTION statement. When a database error statement is added, the generators automatically add the ABORT TRANSACTION statement. These defaults cause the Action Diagram to execute in basically the same way as it would have if the database exceptions had not been added.

You must add your own error-handling statements underneath the appropriate WHEN statement to change this default behavior. You can also delete or change the RETRY and ABORT statements that are added automatically.

Unlike ordinary logical exceptions, such as WHEN not found, a database exception must have at least one action specified for it. Otherwise, the default error-handling procedures are followed for that condition, and the statement is marked as inactive.

**Note:** If the WHEN DATABASE DEADLOCK or TIMEOUT statement is deleted, but the WHEN DATABASE ERROR statement remains, then deadlocks or timeouts that occur during execution are captured by the WHEN DATABASE ERROR condition and follow that path.

## Add Database Exception Logic to an Entity Action Statement

You can add database exception logic to the following entity action statements:

- READ
- READ EACH
- SUMMARIZE
- SUMMARIZE EACH
- CREATE
- UPDATE
- DELETE

- ASSOCIATE (standalone form)
- TRANSFER (standalone form)
- DISASSOCIATE (standalone form)

**To add database exception logic to an entity action statement**

1. Click Analysis or Design, Action Diagram.  
The Action Diagram Names dialog appears.
2. Select a process from the Processes list.
3. Select an action block from the Action Blocks list and click OK.  
The entity action statements are displayed in the Action Diagram window.
4. Right-click the last statement in the entity action statement and click Edit, Add Statement, Database Exception.

The database exception statements are added automatically after each statement.

**Note:** You can also select the statement itself for READ EACH and SUMMARIZE EACH statements. Database Exception logic displays after any other statements that are enclosed by, and subordinate to, the READ EACH or SUMMARIZE EACH. A new "(for each successful iteration)" line also displays immediately before any such subordinate statements. Contract the line "(for each successful iteration)" to display database exceptions next to a READ EACH or SUMMARIZE EACH statement.

5. Type the exception logic you need to add after each WHEN clause.

## ABORT TRANSACTION Statement

The ABORT TRANSACTION statement terminates the execution of the procedure step, regardless if the ABORT TRANSACTION statement occurs within an action block or the procedure step itself. It causes an immediate exit and displays a fatal error message. That message may be either the last error message returned by a database exception or a user-supplied text expression.

In the following example, the last database error message displays:

```
WHEN database error
ABORT TRANSACTION DISPLAYING last database error message
```

In the following example, a user-supplied text expression displays:

```
WHEN database error
ABORT TRANSACTION DISPLAYING concat ("A serious error occurred updating product ",
product id)
```

## Add an Abort Transaction Action Statement

You can use the ABORT TRANSACTION statement after a WHEN DATABASE ERROR statement, as well as, anywhere within an Action Diagram. It is automatically added after a WHEN DATABASE ERROR statement.

### Follow these steps:

1. Click Analysis or Design, Action Diagram.  
The Action Diagram Names dialog appears.
2. Select a process from the Processes list.
3. Select an action block from the Action Blocks list and click OK.  
The entity action statements are displayed in the Action Diagram window.
4. Right-click on the location in the entity action statements after which you need to add action and click Edit, Add Statement, Abort Transaction.  
The Add Statement dialog appears.
5. Complete the expression by choosing from last database error message, char string, character view, function (char), or spaces.
6. Click Add.  
The Abort Transaction statement is added to the action block.

## RETRY TRANSACTION Statement

The RETRY TRANSACTION statement terminates the execution of the procedure step, regardless if the RETRY TRANSACTION statement occurs within an action block or the procedure step itself. It causes an immediate exit, the rolling back of any updates, and the restarting of that procedure step as if that action had never occurred.

The following list details the only differences between the ABORT TRANSACTION statement and the RETRY TRANSACTION statement:

- The system attribute TRANSACTION\_RETRY\_COUNT increments each time the transaction retries.
- The system attribute TRANSACTION\_RETRY\_LIMIT does not reset to its default value at the beginning of the procedure step, but the attribute retains the value it had when the RETRY TRANSACTION statement was executed.



**Note:** If you issue a retry and have already reached the limit of retries, a fatal error message displays. To avoid this condition, follow this example:

```
IF TRANSACTION_RETRY_COUNT < TRANSACTION_RETRY_LIMIT
RETRY TRANSACTION
ELSE
EXIT STATE IS WAIT_AWHILE_AND_TRY_AGAIN
ESCAPE
```

## Add a Retry Transaction Action Statement

You can use the RETRY TRANSACTION statement after a WHEN DATABASE DEADLOCK or TIMEOUT statement, as well as, anywhere in a procedure step or an action block. The RETRY TRANSACTION statement is automatically added to the logic when the WHEN DATABASE DEADLOCK or TIMEOUT statement is added.

### Follow these steps:

1. Click Analysis or Design, Action Diagram.  
The Action Diagram Names dialog appears.
2. Select a process from the Processes list.
3. Select an action block from the Action Blocks list and click OK.  
The entity action statements are displayed in the Action Diagram window.
4. Right-click on the location in the entity action statements after which you need to add action and click Edit, Add Statement, Retry Transaction.  
The Retry Transaction statement is added to the action block.

## REMOVE Statement

The REMOVE statement is part of the UPDATE construct. You can use the REMOVE statement to remove attribute values; however, code generation does not support the REMOVE statement. Use ASSOCIATE, DISASSOCIATE, or TRANSFER to change relationships. Mandatory attributes may not be the subject of a REMOVE clause.

In the following example, the UPDATE action signals modification to the information stored about the PERSON entity in the entity action view PERSON. The SET clause changes the stored name of the PERSON to "John."

```
UPDATE person
SET name TO "John"
REMOVE address
TRANSFER FROM original order WHICH is placed by IT
TO new order WHICH is placed by IT
```

The REMOVE statement eliminates the stored address of the person. The TRANSFER clause replaces the original pairing between the previous PERSON (for whom we do not have a name) and ORDER by a pairing between the former PERSON name (now John) and his new ORDER.

### Inline Code Statement

With the Inline Code statement, CA Gen supports the ability for users to add target-specific source code (COBOL, C, Java, C# and/or SQL) directly into an action diagram. Although this ability is similar to that provided by external action blocks (EABs), with this statement, the code is stored in the model instead of outside the model.

To allow customers to limit the amount of code that may be placed in an individual Inline Code statement, there is a model preference (Model / Model Preferences) in Toolset. This model preference sets the maximum number of characters allowed in any single Inline Code statement in an Action Diagram.

## Call External Statement

Add Call External statements in an action diagram to work with web services. This statement helps you map the parameters of a web service with the views of an entity or work set of your application. You can create entity views on the fly and then map them to the parameters of the web service. You have the option of creating an internal work set whose attributes match the best possible Gen data types with the data types of the parameters.

The action diagram CALL EXTERNAL statement is used to call a web service interface in the following format:

CALL EXTERNAL *<type of external call>* *<method name>*

**<type of external call>**

Specifies the type of external call being made by the statement. One of the types of external call being made is *Web Service*.

**<method name>**

Specifies the WSDL method name.

**For example: The type of external call being made in the following example is Web Service and the method name is GetQuote:**

```
-EXAMPLE_ACTION_BLOCK
|   IMPORTS:
|   EXPORTS:
|   LOCALS:
|   ENTITY ACTIONS:
| CALL EXTERNAL Web Service "GetQuote"
```

## Add Conditional Actions

A conditional action evaluates a situation and directs the flow of a process based on the result. Conditional actions include IF (including ELSE and ELSE IF) and CASE.

Creating conditional actions consists of adding:

- Simple conditions
- Complex conditions
- Conditions dependent on attribute values

The following sections discuss the activities required to build each type of conditional action.

## Add Simple Conditions

You add a simple condition by adding the IF action statement. The IF action specifies the conditions under which process actions take place. In its simplest form, IF allows you to compare two objects. To make simple mutually exclusive conditions, you add ELSE conditions to the IF action statement.

An IF action statement for a simple condition:

```
IF condition_1  
action_statement_list_1
```

The IF action statement for a simple mutually exclusive condition takes the following form:

```
IF condition_1  
action_statement_list_1  
[ELSE  
action_statement_list_2]
```

The actions inside the bracket are performed only if the condition is not true.

Condition 1 is any of the following expressions that can be either true or false:

- Entity view
- Group view
- Explicit expression
- Command
- Exit state
- Current time
- Links
- Current date
- Function
- Special attributes (terminal ID, user ID, printer ID, transaction code)
- Subscript
- Last
- Max

The IF statement merely sets up the condition; it does not indicate the actions to be taken when the condition is met. To complete the description, follow the IF statement with one or more action statements. Since an IF statement contains a comparison, it must contain a relational operator such as IS EQUAL TO or IS LESS THAN. The following example shows an IF statement containing a relational operator:

```
IF processing vendor reference_evaluation
IS GREATER OR EQUAL TO 2
MOVE processing vendor TO export vendor
```

When the IF statement and the ELSE IF condition are not true, conclude the IF statement with an ELSE clause. The actions inside the ELSE bracket are performed only if the condition(s) in the IF statement and all ELSE IF statements are false. For a discussion of ELSE IF actions, see the next section. When you select ELSE, you can add one or more actions to the ELSE block.

The following statement illustrates the use of ELSE with IF:

```
IF (processing vendor average_delivery_time_in_days)-
(competitive vendor average_delivery_time_in_days
IS EQUAL TO 0 AND processing vendor reference_
evaluation IS EQUAL TO competitive vendor
reference_evaluation
IF competitive vendor average_delivery_time_in_days
IS LESS OR EQUAL TO 5
MOVE competitive vendor TO preferred vendor
ELSE
MOVE competitive vendor TO ok vendor
```

You can also use simple conditions with repeating group views. The following statement illustrates a simple condition that references a subscript and sets a subscript in a repeating group view:

```
IF SUBSCRIPT OF export_list > MAXIMUM OF export_list
SET SUBSCRIPT OF export_list TO MAXIMUM OF export_list
SET output customer name TO "Press F5 for More"
```

You can add an IS VALID expression to an IF condition to check for permitted values. To test whether persistent views can be used for UPDATE, DELETE, ASSOCIATE, or DISASSOCIATE actions, you can add the following expressions:

- IS POPULATED
- IS NOT POPULATED
- IS LOCKED
- IS NOT LOCKED

The expressions IS POPULATED and IS NOT POPULATED test whether a READ statement populated a view. The expressions IS LOCKED and IS NOT LOCKED test whether the view is still locked from a previous READ and may be updated without another READ statement.

Successful READs and CREATEs set the POPULATED flag on. Unsuccessful READs and DELETEs set the flag to off. The flag can be checked with the IF POPULATED / IF NOT POPULATED statements. These statements should be used in Action Diagrams accepting persistent views. If the populated flag is off, a subsequent UPDATE would cause a runtime error, TIRM039E. To protect against this, have a path in your Action Diagram wherein a READ is performed if the Populated flag is off.

If a previously populated view is involved in a subsequent unsuccessful READ, the populated flag is turned off, but the data remains in the view. This can cause more than a little confusion when displaying views using CA Gen Diagram Trace Utility.

The Locked/Unlocked parameter is only for communication during code generation. As a result, something else must ensure at runtime that a lock is established when the used action block is executed. The IF LOCKED/IF NOT LOCKED statements are used to make this test. If the view is not locked and needs to be, you must READ the entity occurrence again or a TIRM039 runtime failure occurs with a DU status code.

### Add Complex Conditions

You can add complex mutually exclusive conditions by adding any number of ELSE IF clauses to an IF action statement, followed by the ELSE clause. You can have a series of actions in which only one may take place based on the satisfaction of the corresponding ELSE IF condition.

The IF action statement that represents a complex mutually exclusive condition takes the following format:

```
IF condition_1
action_statement_list_1
[ELSE IF condition_2
action_statement_list_2]...
[ELSE
action_statement_list_3]...
```

The following example of an IF statement contains a complex mutually exclusive condition that tests the rating of a vendor:

```
IF processing vendor company_rating IS EQUAL TO "BEST"
MOVE processing vendor TO preferred vendor
ELSE IF processing vendor company_rating IS EQUAL TO
"FAIR" OR processing vendor company_rating IS
EQUAL TO "GOOD"
MOVE processing vendor to ok vendor
ELSE
MOVE processing vendor TO export vendor
```

### Add Conditions Dependent on Attribute Values

Adding conditions dependent on attribute values consists of building a CASE OF and subordinate CASE action statements.

The CASE action statement appears in the following format:

```
CASE OF expression _1
CASE constant_1
action_statement_list_1
[CASE constant_2
action_statement_list_2]...
[OTHERWISE
action_statement_list_3]...
```

The CASE OF conditional action specifies actions that depend on the value of a specific attribute. All actions depend on the same attribute and are mutually exclusive.

To add conditions dependent on attribute values, begin by specifying the attribute whose value determines the action. Specify each CASE attribute value as shown in the following example:

```
CASE OF competitive vendor company_rating
CASE "BEST"
CASE "GOOD"
OTHERWISE
```

Then, specify the action for each possible value of that attribute, as shown in the following example:

```
CASE OF competitive vendor company_rating
CASE "BEST"
MOVE competitive vendor TO preferred vendor
CASE "GOOD"
MOVE competitive vendor TO ok vendor
OTHERWISE
```

Finally, specify what happens when the attribute value is equal to none of the above using OTHERWISE. This is shown in the following example:

```
CASE OF competitive vendor company_rating
CASE "BEST"
MOVE competitive vendor T0 preferred vendor
CASE "GOOD"
MOVE competitive vendor T0 ok vendor
OTHERWISE
MOVE competitive vendor T0 export vendor
```

When you implement a process in the Dialog Flow Diagram, CA Gen automatically creates a Procedure Action Diagram containing a CASE statement. When you specify commands during process-to-procedure implementation, the CASE statement includes a USE action for each command selected. This is shown in the following example:

```
CASE OF COMMAND
CASE modify
USE modify_purchase_order
CASE log
USE enter_purchase_order
CASE cancel
USE cancel_purchase_order
OTHERWISE
```

**Note:** For more information about the CASE command, see the *Design Guide*.

## Add Repeating Actions

A repeating action supports the repetition of a set of action statements under different conditions. Repeating actions include READ EACH, FOR EACH, WHILE, REPEAT UNTIL, and FOR.

Adding repeating actions, part of defining logic for elementary processes, consists of three activities:

- Add READ EACH action
- Add FOR EACH action
- Add repeat conditions

### Add READ EACH Entity Action

Adding READ EACH entity actions to the Action Diagram is made up of the following subordinate activities:

- Add READ EACH entity action
- Specify desired group view



- Target group view
- Specify sorting of READ results, if desired
- Specify any selection criteria
- Detail properties of a READ EACH statement

A READ EACH entity action retrieves information about multiple entities of a given type that meet specified selection criteria. The results of the READ EACH and its MOVE action can populate a repeating group view.

The DBMS dictates how the entity occurrences and their attribute values are retrieved when the READ EACH is first encountered during an execution. The DBMS does not indicate whether it retrieves data individually or all occurrences at once. The individual case is more common. If you change an entity occurrence after that, the change is not reflected in the next iteration.

**Downstream Effects:** READ EACH statements always cause a cursor to be used in the DBMS. For DB/2 mainframe batch applications, READ EACH statements can be generated with cursor hold. To access the option, select the Read Each statement. Select the Detail pull-down and the Properties action.

**Note:** For more information, see the Toolset Help.

The READ EACH action uses the following format:

```
READ EACH entity-view-list
[TARGETING repeating-group-view-1 [FROM THE BEGINNING]
[UNTIL FULL]]
[AND TARGETING repeating-group-view-1 [FROM THE BEGINNING]
[UNTIL FULL] . . .
[SORTED BY ASCENDING attribute 1] . . .
DESCENDING
[WHERE selection-conditions]
action-statement-list
```

The first line indicates the READ EACH action and the entity action views(s) being read. Entity-view-list indicates that you can specify multiple entity action views in the READ EACH statement. For a discussion of the advantages of reading multiple views in one READ statement, see the Add READ Entity Action section; the same advantages pertain to READ EACH statements.

The next four lines target the group view. A group view is a collection of entity views, other group views, or both. Only group views can be identified as repeating. The four lines contain optional TARGETING clauses that target repeating group views to be populated each time the READ EACH repeats. The repeating group views may be either local or export views. The TARGETING clause allows you to identify (connect) the repeating group read with the repeating export or local group that may be populated. The clause establishes the sequence in which the occurrences of the entity views within a repeating export or local group are populated. You can add multiple TARGETING clauses to allow multiple repeating group views to be populated simultaneously.

Part of the TARGETING clauses include the FROM THE BEGINNING and UNTIL FULL clauses. The FROM THE BEGINNING clause is useful if multiple READ EACH actions target the same repeating group view. To begin populating the group with the first entity view in the group, specify FROM THE BEGINNING. If you do not specify FROM THE BEGINNING, the group is populated beginning with the current entity view. The UNTIL FULL clause causes the READ EACH to terminate if an attempt is made to exceed the maximum number of occurrences set for the repeating group view.

You specify the order of the READ with two clauses: SORTED BY ASCENDING and SORTED BY DESCENDING. The SORTED BY clause returns entities in a particular sequence based on an attribute value. The ASCENDING clause returns the entities in a low-to-high order. The DESCENDING clause returns the entities in a high-to-low order.

The optional WHERE clause indicates selection conditions used for the READ EACH action. Refer to Specify Selection Conditions for a READ EACH Action.

Action-statement-list indicates a block of actions to occur for each occurrence.

One iteration occurs per distinct combination of entity occurrences that satisfy the selection criteria. One occurrence can be retrieved for more than one iteration if other view(s) in the read list retrieve new occurrences. Every view is populated for every iteration. If any view cannot be populated for a given combination of occurrences, then that combination is rejected (no iteration).

The following examples illustrate READ EACH statements (action-statement-list is not shown):

```
READ EACH product
WHERE DESIRED product code IS EQUAL TO "GA"
READ EACH product
TARGETING group export FROM THE BEGINNING UNTIL FULL
WHERE code IS EQUAL TO "GA"
customer_order
READ EACH customer order line
SORTED BY customer_orderline_number ascending
WHERE DESIRED CUSTOMER_ORDER_LINE_PART_OF_CURRENT
```

## Specify Selection Conditions for a READ EACH Action

Specifying selection conditions for a READ EACH entity action is part of adding the action. Specifying selection conditions consists of the following activities:

- Specify the WHERE expression
- Specify attribute conditions
- Specify relationship conditions

The selection criteria is specified in terms of attributes, relationships and combinations thereof. When specifying a combination of attributes and relationships, specify the attributes first.

READ EACH statements can be simple or complex. For either simple or complex statements, CA Gen makes statement construction easy by presenting only valid choices.

A selection condition takes the following form:

```
[ ( ) attribute-condition-1 [ ] ] AND [ ( ) attribute-condition-2 [ ] ]  
or  
OR  
relationship-condition-1 & OR relationship-condition-2  
or  
AND
```

## Specify Attribute Conditions for a READ EACH Action

The same attribute conditions apply to the READ and the READ EACH actions. For information on the syntax and types of expressions used to specify attribute conditions for a READ EACH action, see the Specify Attribute Conditions for a READ Action.

The following example shows how READ EACH attribute conditions are used in Action Diagrams. The first example discovers whether the business knows about the CUSTOMER who is attempting to place an ORDER (in the process Take Order):

```
READ EACH customer  
WHERE DESIRED customer name IS EQUAL TO requesting  
customer name
```

The next example reads all customers whose name begins with Q:

```
READ EACH customer  
WHERE SUBSTR(DESIRED customer name, 1,1) IS EQUAL TO  
"Q"
```

**Note:** If the qualifying attributes of a READ EACH action must be changed before subsequent iterations of the READ EACH are executed, an escape could be placed within the READ EACH. The qualifying attributes could then be set to new values, and those set statements along with the READ EACH could be placed within a REPEAT loop.

## Specify Relationship Conditions for READ EACH

Like attribute conditions, the same relationship conditions apply to the READ and the READ EACH actions.

The following examples show how relationship conditions are used in Action Diagrams. The first example reads each CUSTOMER ORDER pair where the customer placed the order for the current occurrence of PRODUCT; it also reads any ORDER(s) placed for that PRODUCT:

```
READ EACH customer
order
WHERE DESIRED customer places DESIRED order
AND DESIRED order contains SOME order line
AND THAT order_line is for CURRENT product
```

The next example shows a fully defined READ EACH statement that targets a repeating group view:

```
READ EACH issue
status
TARGETING out_group FROM THE BEGINNING UNTIL FULL
SORTED BY ASCENDING issue number
WHERE DESIRED issue_status is_contained_in
CURRENT design_development
AND (DESIRED issue_status status IS EQUAL TO "Q"
OR DESIRED issue_status status IS EQUAL TO "T"
OR DESIRED issue_status status IS EQUAL TO "N")
MOVE issue_status to output issue_status
MOVE issue TO output issue
```

### More information:

[Specify Relationship Conditions for a READ Action](#) (see page 24)

## Detail Properties of a READ EACH Statement

You can specify whether a DISTINCT clause is generated in a READ EACH statement using the READ EACH Properties dialog. The dialog also allows you to specify whether an OPTIMIZE FOR clause is generated for DB2 applications.

**Note:** For more information, see the Toolset Help.

To access the dialog and Toolset Help, select the READ EACH statement. With the statement highlighted, select the Detail pull-down menu and the Properties action.

To select this option, display the READ EACH properties dialog by either double clicking on the READ EACH statement or by selecting the READ EACH statement and then selecting Detail—>Properties from the menu bar. On the properties dialog, click the "Generate with cursor hold" check box.

This option is valid only for DB2. This option is ignored for all other DBMSs.

## Add FOR EACH Condition

The FOR EACH condition is used to iterate through implicitly indexed repeating group views. Since the view is implicitly indexed, there is no need to specify subscript controls. This is automatically generated by CA Gen software.

CA Gen presents a list of implicitly indexed group views when you select FOR EACH. You cannot use FOR EACH if there are no implicitly indexed repeating group views.

The TARGETING clause allows you to identify the repeating export or local group view to be populated. The following example shows usage of the TARGETING clause:

```
FOR EACH product details
TARGETING group export FROM THE BEGINNING UNTIL FULL
ASSOCIATE processing warehouse
WITH received product WHICH is_held_in IT
```

**Note:** For a description of the TARGETING clause, see the description of READ EACH in Add READ EACH Action.

After you add the FOR EACH statement, add the actions to be repeated for each occurrence of the entity.

**Note:** For information on specifying selection criteria for a FOR EACH statement, refer to the selection conditions for a READ action.

### More information:

[Add READ EACH Entity Action](#) (see page 64)

## Add Repeat Conditions

Adding repeat conditions consists of the following activities:

- Add WHILE condition
- Add REPEAT UNTIL condition
- Add FOR condition

## Add WHILE Condition

The WHILE repeat condition repeats the action group as long as a condition is true. When you add the WHILE condition, set up the condition and then state the action(s) to be taken. The test for the condition takes place first. If the condition is not met, there are no iterations.

The WHILE statement can contain multiple TARGETING statements that target repeating group views.

The WHILE statement uses the following format:

```
WHILE condition-1
[TARGETING repeating-group-view-1 [FROM THE BEGINNING]
[UNTIL FULL]]
[AND TARGETING repeating-group-view-1 [FROM THE BEGINNING]
[UNTIL FULL]] . . .
action-statement-list
```

The following example of a WHILE statement sets a condition for a product code and USEs the action block *check against order*:

```
WHILE received product code IS EQUAL TO "WI"
USE check against order
WHICH IMPORTS: Entity View received product
WHICH EXPORTS: Entity View received product
```

## Add REPEAT UNTIL Condition

The combination of the REPEAT and UNTIL statements repeats an action or group of actions until a specific condition is satisfied. The test for the condition takes place after the iteration. Therefore, REPEAT-UNTIL always has at least one iteration.

The REPEAT-UNTIL statements can contain multiple TARGETING statements that target repeating group views.

The REPEAT-UNTIL statement uses the following format:

```
REPEAT
[TARGETING repeating-group-view-1 [FROM THE BEGINNING]
[UNTIL FULL]]
[AND TARGETING repeating-group-view-1 [FROM THE BEGINNING]
[UNTIL FULL]] . . .
action-statement-list
UNTIL condition-1
```

The following example of REPEAT–UNTIL statements repeats the referenced business algorithm *find space* until a value is met.

```
REPEAT
USE find space
WHICH IMPORTS: Entity View processing warehouse
WHICH EXPORTS: Entity View processing warehouse
UNTIL processing warehouse number_of_available_bins IS EQUAL TO 0
```

The following REPEAT–UNTIL statements use a FOR EACH action:

```
REPEAT
FOR EACH product details
ASSOCIATE processing . . .
WITH received . . .
UNTIL. . .
```

Finally, the following example targets the group view *order\_line\_export*:

```
REPEAT
TARGETING order_line_export FROM THE BEGINNING UNTIL FULL.
UNTIL . . .
```

## Add FOR Condition

The FOR Action Diagram statement provides a concise way to set up a program loop. It is often used to control iterating through an explicitly indexed repeating group view. The syntax allows you to pick a subscript or local numeric attribute view, a starting value, an ending or limit value, and an increment value for each iteration.

The FOR Action Diagram statement is similar to the FOR statement that is provided by many programming languages. However, one major difference is that any changes to the limit and increment values during iteration of the FOR loop will not affect the actual incrementation or limit testing that is done. This is because these values are saved in work variables before the loop begins, and those work variables are used for each iteration to avoid re-evaluating the limit and increment expressions each time.

Another difference between the FOR Action Diagram statement and a typical FOR statement in a programming language is that, if the increment value in a FOR Action Diagram statement is negative, the loop will terminate when the control view or subscript is less than or equal to the limit or ending value. Otherwise, the loop will terminate when the control view or subscript is greater than or equal to the limit or ending value.

In the following example, the FOR action statement starts a loop that repeatedly executes the IF statement that follows:

```
FOR local work_items counter FROM 1 TO 10 BY 1
  IF input customer_order confirmed_date IS LESS OR EQUAL TO CURRENT_DATE + local
  work_items counter DAYS
  EXIT STATE IS confirm_date_order
<--ESCAPE
```

The first time through the loop, the IF statement is executed with a value of 1. If the conditions are met, the local counter increments by 1 and the entire set of statements executes again. The process repeats 10 times. If the conditions are not met (input *customer\_order confirmed\_date* is greater than the current date plus the number of days indicated by the counter), the exit state is set and the FOR loop is terminated by the ESCAPE statement.

The following FOR statement example shows how a reference to the subscript of an explicitly indexed repeating group view appears in the Action Diagram:

```
FOR SUBSCRIPT OF group_customer_order
FROM 1 TO MAX OF group_ customer_order BY 1
```

The following example of a fully defined FOR statement uses an explicitly indexed repeating group view. The statement shows how subscripts are used both to control the FOR loop and to calculate totals.

```
FOR SUBSCRIPT OF loan_payments FROM 1 TO 360 BY 1
  SET payment principle TO (loan principle amount * table factor)
  SET payment total TO payment principle + payment interest
  SET local work_view yearly_total TO local work_view
  yearly_total + payment total
  IF local work_view month_counter IS EQUAL TO 12
    SET yearly payment total TO local work view yearly_total
    SET local work_view month_counter TO 1
    SET SUBSCRIPT OF yearly_totals TO yearly totals + 1
  ELSE
    SET local work_view month_counter TO 1
```

The following example shows how expressions may be used as the initial, increment, and limit values:

```
FOR local employee fica_withheld_ytd FROM (employee salary * employee fica_rate) TO
(employee salary * 12 * employee annual_fica_max_rate) BY (employee salary * employee
fica_rate)
IF local employee fica_withheld_ytd IS GREATER THAN (employee salary * 12 * employee
annual_fica_max_rate)
SET local employee fica_withheld_ytd TO (employee salary * 12 * employee
annual_fica_max_rate)
```



In the preceding example, the loop will terminate when the `employee_salary * 12 * employee_annual_fica_max_rate` value is reached. The enclosed IF block prevents local employee `fica_withheld_ytd` from exceeding it.

The following example shows that a change to the limit or increment during execution of the FOR loop will not affect the values actually used during execution of the FOR statement:

```
SET local work_items limit TO 10
FOR local work_items counter FROM 1 TO local work_items limit BY 1
    SET local work_items limit TO 5
```

When the preceding FOR statement terminates, local `work_items` counter will have a value of 10, not 5, even though the limit variable was reset during the loop. This is because the values of the limit and increment expressions are saved in work variables before the first FOR iteration, and the saved values are the ones actually used during the execution of the statement.

## Add Control Actions

A control action changes the flow of an elementary process unconditionally. Control actions specify the reuse of other action groups and provide a structured means of terminating the execution of action groups.

Adding control actions consists of five activities:

- Add USE actions
- Add Procedure Step USE actions
- Add NEXT actions
- Add ESCAPE actions
- Add ASYNC actions

The following topics discuss these activities.

### Add ASYNC Action

A set of Async action statements are used to invoke logic in an online, non-display Procedure Step asynchronously to the ongoing execution of the invoking action block.

#### More information:

[Asynchronous Behavior](#) (see page 171)

## Add USE Action

The USE action allows you to call other action blocks from procedure and action blocks. The called action block cannot be a Procedure Step or a derivation algorithm.

In the Action Diagram, after selecting USE, you select the action block to be used. You then match import views, export views, local views, and entity action views.

The following example shows how a USE statement references an action block called *find\_space*: The views in the WHICH IMPORTS list populate the import view of the action block before it begins execution. The views in the WHICH EXPORTS list are populated from the export view of the action block on return.

```
USE find_space
WHICH IMPORTS: Group View product information
Entity View storage_space warehouse
WHICH EXPORTS: Entity View used warehouse
```

The following example shows how a USE statement references an action block called *create\_order*:

```
USE create_order
WHICH IMPORTS: Group View group_import Entity View import customer Entity View import
customer_order
WHICH EXPORTS: Group View group_export Entity View export customer Entity View
customer_order
```

In the following example, another form of the USE action SETs the value of an attribute USING an algorithm:

```
SET employee_number USING number_generator
```

After you select the algorithm, CA Gen prompts you to match the import views of the algorithm with the supplying views from the process.

## Add Procedure Step USE Actions

A Procedure Step USE statement calls logic in an online, non-display Procedure Step.

**Note:** For more information, see the Toolset Help.

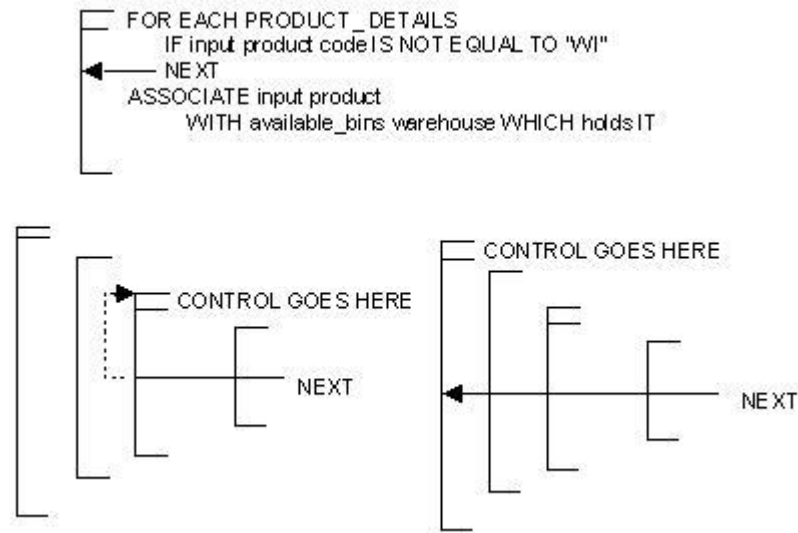
## Add NEXT Action

The NEXT action is used within repetitive action statements like WHILE and FOR to go directly to the next iteration without completing the current one.

NEXT allows you to perform an action on some but not all entities TARGETED in a repetitive action. For example, in the following statement, you are interested only in input products with code *WI*.

**Note:** The head of the arrow in the NEXT action shows which action group terminates and which action group gains control. The second part of the diagram shows how you can specify control to other action groups with the NEXT action.

This diagram shows the Add NEXT Action:



You want to ASSOCIATE those input products with an available bin in a warehouse. If you find a code other than W/ you do not want to stop the entire process; you simply do not want to ASSOCIATE that product with a warehouse bin.

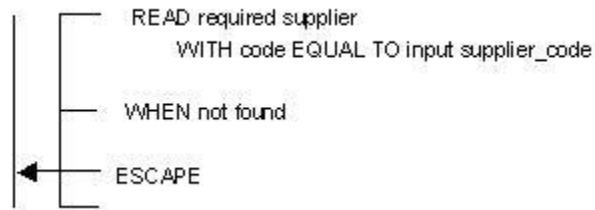
## Add ESCAPE Action

The ESCAPE action terminates the execution of a particular action group. ESCAPE is frequently used after exception conditions such as WHEN already exists and WHEN not found.

ESCAPE is just like the GOTO statement that appears in many languages, and has all the drawbacks of the GOTO, in that it can make your Action Diagrams difficult to read and debug.

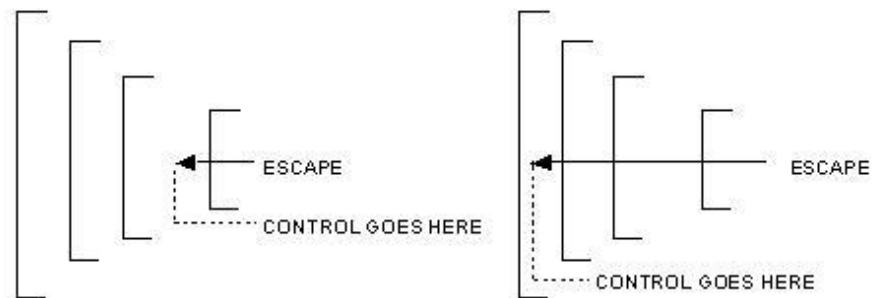
The following example shows how ESCAPE terminates the READ of REQUIRED SUPPLIER and specifies which action group to process next:

This is an example of ESCAPE:



The head of the arrow in the ESCAPE shows which action group terminates and which action group gains control. The following diagram shows how you can specify control to other action groups with the ESCAPE action.

This is an example of ESCAPE Action Statement:



## Add Miscellaneous Actions

Defining logic for an elementary process and a procedure can include adding miscellaneous actions. Adding miscellaneous actions consists of three activities:

- Add MAKE action
- Add NOTE action
- Add a blank line

### Add MAKE Action

The MAKE action dynamically sets the video properties of on-screen fields from an Action Diagram. The MAKE action has meaning only in online procedures.

**Note:** The MAKE command is available for selection only when you select a Procedure Step. To access a Procedure Step, select Design, Action Diagram, which is a Procedure Action Diagram. CA Gen software does not allow you to add MAKE actions in the Process Action Diagram or action blocks.

The MAKE statement should be used carefully to assure that your system color scheme is followed.

The following example uses the MAKE action to set the video attribute to the default ERROR values:

```
MAKE ... ERROR
SET Video Attributes
```

This is the preferred statement for setting video attributes.

The following example applies the MAKE action to set video attributes:

```
MAKE input purchase_order_status Unprotected Normal
intensity
Pink Underscored
MAKE output division number
Unprotected HIGH Intensity Yellow Underscored
```

Notice that this last example might violate your local Screen Design standards.

## Add NOTE

The NOTE action allows you to document your diagram. You can insert a NOTE anywhere in the Action Diagram. When you select NOTE, enter the text in the pop-up window.

The following example of NOTE action conveys information about the purpose of a procedure used in a general ledger system.

**Note:** The purpose of this procedure is to display the requested journal entry header and its associated line items.

## Add a Blank Line

A blank line between actions can make a diagram easier to read. You can add a blank line anywhere except inside an entity action statement.

## Define Action Blocks

An action block is a named collection of action statements that do not directly support an elementary process. A common action block is an action block invoked by more than one process or procedure. A business algorithm is an action block invoked through a USE action or a SET USING action.

Defining logic for action blocks consists of the following activities:

- Create new action block in Analysis or Design
- Define logic for action blocks
  - Define algorithm for derived attributes
  - Define algorithm for designed attributes
- Reference common action blocks in process and Procedure Action Diagrams
  - Specify business algorithm in SET USING statements
  - Specify business algorithm in USE statement
- Define external action blocks
  - Add EXTERNAL designation
  - Define import and export views

The following topics discuss these activities.

### Create New Action Block in Analysis or Design

You must create a new action block before you define it. You can create a new action block by assigning a name to a new action block or by copying an existing action block with substitution (referred to as copy with substitution).

### Define Action Blocks for Processes and Procedures

When you build action blocks for elementary processes and procedures, you define algorithms for both derived and designed attributes. The same logic rules apply to both Action Diagrams and action blocks. The Action Diagram displays the actions and expressions you previously encountered in building an Action Diagram.

## Reference Common Action Blocks

Common action blocks see blocks of logic that more than one process or Procedure Action Diagram can reference through SET USING and USE actions. Existing action blocks or diagrams can USE a defined action block. The business system in which you create or scope the action block does not own it.

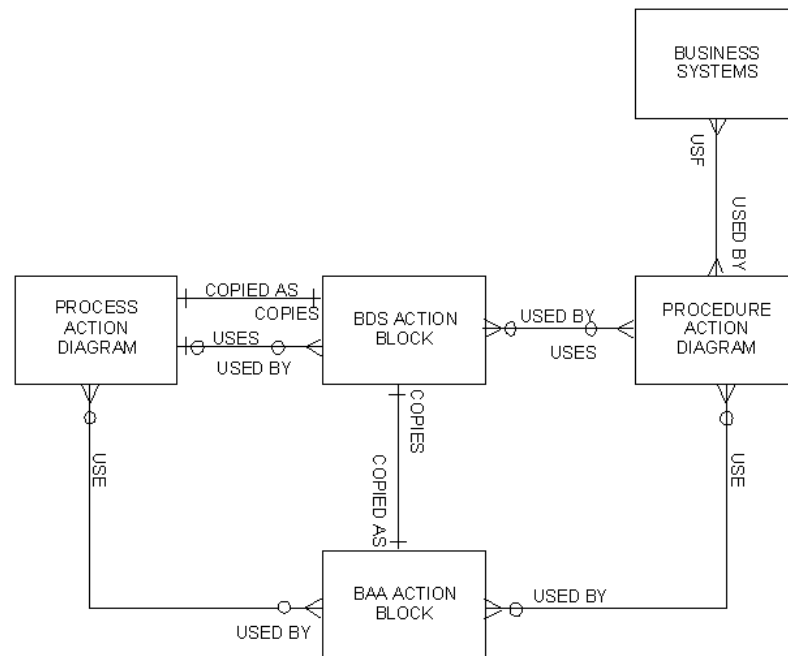
When you create a new elementary process in the activity model, the process appears in two places:

- As an Analysis Process Action Diagram
- As a Design action block

When you create a new action block in Analysis or Design, the action block appears in two places:

- As an Analysis action block
- As a Design action block

The following illustration shows Relationships between Action Blocks and Action Diagrams in Business Systems:



## Define External Action Blocks

External action blocks that you create with the Action Diagram tool define the interface between Procedure Steps (or action blocks) and subroutines created outside CA Gen. The external action block provides a structure to match the views of the Procedure Step with the views (input and output arguments) of the handwritten subroutine.

An external action block contains only import views, export views, and an EXTERNAL designation, as shown in the following example:

```
READHIST
IMPORTS: Entity View new customer
EXPORTS: Entity View existing customer
EXTERNAL
```

CA Gen Procedure Step or action block that uses the external action block supplies the import views. The Procedure Step or action block calls the external action block through the USE control action. The import views define the data passed from the CA Gen-generated application to the interface routine. The export views define the data returned to the generated application from the interface routine.

When an EXTERNAL action block is generated, only the views structure and a program shell are built. You must supply any program logic required by your application.

**Note:** For more information about external action blocks, see the *Workstation Construction User Guide*.

## Changing the Action Diagram

The Action Diagram tool allows you to change statements after they are added. Just as when you are adding the statements, only syntactically valid choices are presented to you as you request the change.

### Complex Changes

Complex changes apply to complex action statements. For complex changes, CA Gen lets you add new text, add parenthetical expressions, and delete parts of some statements.

CA Gen prevents you from performing the following activities when you make complex changes:

- Deleting invalid combinations of expressions within a statement
- Changing a statement that has associated statements



## Perform Simple Changes

Simple changes generally consist of replacing a component in the Action Diagram with the same type of object. Unlike complex change, which allows you to change many parts of an action group, simple changes map one-to-one. Exceptions are expressions: when you change an expression, CA Gen lists valid choices.

You can make simple changes to views, expressions, operators, note descriptions, and exit state values.

## Copying an Action Diagram

The Action Diagram tool allows you to copy procedures, Procedure Steps, action blocks, and Action Diagrams. All of these can be copied exactly, or copied with substitution.

When copying exactly, the result is two identical objects with different names.

When copying with substitution, you can specify different entity types, attributes, and relationships for those in the Action Diagram that is being copied.

**Note:** For more information, see the Toolset Help available in the Action Diagram tool.



# Chapter 3: Event Processing and GUI Statements

---

Applications with a Graphical User Interface (GUI) allow application users to select a push button, click on an item, or interact with an application in several other ways. These actions can trigger logic that opens a dialog box, highlights an item, or initiates additional logic.

The actions of the application user are called events. The logic triggered by an event is contained in an event action, a special block of logic at the end of a Procedure Step.

A common event that can trigger an event action occurs when an application user clicks OK. This event can trigger an event action that contains the logic to set an exit state that causes a dialog flow.

After an event action executes:

- Local and entity action views are cleared
- Database commits/rollbacks take place
- Exit states are interrogated
- Export views are displayed
- EVENT ACTION event name

## Adding Event Actions

Event actions can be added to a Procedure Step using the Action Diagram or the Window Design functionality within the Navigation Diagram. Use either tool to assign the Event Action Name.

In the Window Design functionality, select a window or dialog and select Detail, Events.

In the Action Diagram, select a Procedure Step. Select Edit, Add Event Action.

Use the Window Design functionality to associate the event action with an event type.

**Note:** For more information, see the *Design Guide*.

## Event Action Names

The name of an event action must be unique in the Procedure Step.

In the Window Design functionality, CA Gen automatically supplies an event action name when an event is added and no name is supplied. The following convention is used:

```
window or dialog box name_[control type]_[object name] event
[control type] is pb (push button), ef (entry field), etc.
[object name] is the name of the push button, entry field, etc.
```

## Event Types

Each event action must be associated with one or more event types.

The event type classifies an event as one that opens or closes dialog boxes, validates changed data, repopulates a scrollable list, or responds when an application user clicks or double clicks on an item. Additional event types can be added using the Window Design functionality.

## CA Gen Supplied Event Types

All event types supplied by CA Gen can be triggered when an application user directly affects a window or a control. These event types include:

- Open and Close
- Changed
- Click and Double-click
- ScrollTop and ScrollBottom
- Gain and Lose Focus
- Activated and Deactivated
- LeftMouseButtonDown and LeftMouseButton Up
- RightMouseButtonDown and RightMouseButtonUp
- MouseMove
- MouseEnter and MouseExit
- WinMove
- WinResize
- Keypress

**Note:** For more information about user-defined event types, see the *Design Guide*.

## User-Defined Event Types

User-defined event types do not require the direct action of an application user. Instead, these event types can be triggered by TIREVENT, a special external action block.

TIREVENT is pre-defined to the runtime components. You do not have to compile it outside of CA Gen or write any other code.

The example that follows illustrates how TIREVENT must be defined in a Procedure Step.

```
TIREVENT
IMPORTS:
Work View IEF_Supplied (transient, optional, import only)
command (required)
EXPORTS:
LOCALS:
ENTITY ACTIONS:
EXTERNAL
```

TIREVENT must be used in the Procedure Step as follows:

```
SET local IEF supplied command to <your_event_type>
USE TIREVENT
WHICH IMPORTS: local IEF supplied command
```

The name of the user-defined event type is passed to the TIREVENT external action block (your\_event\_type). The event action name IS NOT passed to TIREVENT. Both names are entered using the Events dialog of the Navigation Diagram.

**Note:** For more information about user-defined event types, see the *Design Guide*.

## Action Diagram Statements for GUI Applications

GUI statements, statements with dot notation, and GUI object references can be added only to a Procedure Step or to an event in a Procedure Step. They cannot be added to action blocks or Process Action Diagrams.

GUI statements can manipulate dialog, items in a list box, push buttons, and menu items.

- Window and Dialog Statements

- Open
- Close
- Refresh

**Note:** In AllFusion Gen 7, an Open Dialogbox statement in a Close Event will cause the Dialogbox to be opened and displayed. In prior releases, if an Open Dialogbox statement was encountered within a Close Event, the Dialogbox was not displayed. If you have such an Open Dialogbox statement in a window Close Event and you want the original window to remain open when the Dialogbox is closed, you must explicitly provide an Open Window statement. This will cause the Close action to be aborted. Otherwise, the original window will be closed.

- Statements for Push buttons and Menu Items

- Enable Command
- Disable Command
- Mark Command
- Unmark Command

- List Box Statements

- Sort
- Filter
- Unfilter
- Display
- Highlight
- Unhighlight
- Add Row
- Remove Row

- Get Row

## GUI Object Domain

An attribute of GUI Object domain is an interface pointer to an object. In keeping with the object-oriented nature of OLE, an application does not have access to the entire object, just to its external interfaces. These external interfaces are called methods, properties, and events.

The object could be a standard window control such as a button, listbox, or entry field, or an embedded/linked object such as a Microsoft Excel™ spreadsheet or Word™ document.

Typically in application logic, a GUIObject attribute is used to temporarily reference various objects as required to interact with those objects. It is possible for a GUIObject to contain a reference to an object even though that object no longer exists. When application logic sets a GUIObject attribute to a new reference, any existing object referenced by that GUIObject is first released before being set to the new value. When a GUIObject attribute contains an object reference and that object still exists and has not been destroyed, the GUIObject can successfully be set to a new value without first setting the GUIObject to NOTHING. But, if that initial object no longer exists, then the release of that reference will follow an invalid pointer to a destroyed object.

## Dot notation

Dot notation is a syntax used by many development programs today. When implementing dot notation, you use periods or "dots" to separate each part of the expression from the next.

The benefits of dot notation in Procedure Action Diagrams are:

- Reduces number of views that are created
- Reduces logic that is written
- Reduces time needed to create an application





# Chapter 4: Introduction to Views

---

This chapter begins the discussion of views, which includes how to create, use, and manipulate views in specific tools. These activities span tasks that are associated with Analysis and Design tools.

To understand views, you must first understand the relationship between entities and processes. By itself, an entity has little meaning to a business. An entity takes on meaning when a business activity processes information about the entity and makes that information available. For example, unless a business knows the name, address, and telephone number of a customer named Jane Doe, the fact that she is a customer is of no interest to the business. The mechanism through which an activity (generally an elementary process) receives, uses, and produces information is a view. It is a window through which a process communicates information about entities.

Another way of illustrating views is as vantage points from which processes *see* entities. You provide views on a need-to-know basis. For example, one process might need to view only the Status attribute of an entity occurrence of CUSTOMER to accomplish its task. Another process might need to see the Name and Address attributes of CUSTOMER. By carefully defining the scope of views, you can control and restrict access to shared information.

## Views

You create views through view maintenance by assigning a view set, or a collection of views, to a process. In the view set, you specify three types of views that processes can relate to entities:

- Import views show information about an entity that a process receives (or imports)
- Export views show information about the entity that a process produces (or exports)
- Entity action views show stored information about an entity that a process manipulates during execution

Import, export, and entity action views are called view subsets, meaning possible groupings of views within a view set. A fourth view subset, local views, includes views in which processes temporarily save information during execution.

The following illustration shows the types of views you can define at each stage of the methodology and the tools used. The following chart allows you to view subsets by activity and tool.

Activity	Tool	ViewSubset			
		Import	Export	Local	Entity Action
Planning (see note)	AHD	X	X		
	ADD	X	X		
Analysis	AHD	X	X		
	ADD	X	X		
	AD	X	X	X	X
	SC	X	X		
	ABU	X	X		
Design	DLG	X	X		
	SD	X	X		
	AD	X	X	X	X
	SC	X	X	X	X
	ABU	X	X	X	X

AD = Action Diagram

ABU = Action Block Usage

ADD = Activity Dependency Diagram

AHD = Activity Hierarchy Diagram

DLG = Dialog Design

SC = Structure Chart

**Note:** You can create views for functions during Planning but, in practice, you rarely need to specify this type of information at the function level. You may wish to record views for functions if you anticipate some benefit for doing so. Generally you only record views for processes and, most frequently, for elementary processes.

## Using Views

After you create views, you use them in specific tools. In the Activity Dependency Diagram, you associate views of a process with an external object. In the Dialog Design, you automatically invoke view synthesis during process-to-procedure implementation. You also conduct view matching to match views passed between Procedure Steps.

In the Action Diagram, you conduct view matching to match views between processes and Procedure Steps and the action blocks they call.

**Downstream Effects:** The following discussion describes how views are used throughout all stages of Information Engineering, starting with their creation in Analysis and ending with application generation in Construction.

During Analysis, view creation actually starts when you add entity types and subtypes, and their attributes to the data model with the Data Model tool. These entity types and subtypes with their attributes are then assigned to import and export views with the Activity Hierarchy and Activity Dependency Diagram tools. The Process Action Diagram automatically reflects the import and export views you defined for elementary processes with the Activity Hierarchy and Activity Dependency Diagram. If needed, you can add any missing import and export views using the Action Diagramming tool. You can also add entity action and local views.

During Design, you specify procedures with the Dialog Design tool. You implement one or more elementary processes in procedures. During transformation, CA Gen automatically synthesizes a set of views based on the combined views of the processes being implemented and places the views in a Process Action Diagram.

The Process Action Diagram uses the following views:

- Synthesized import and export views from elementary processes
- Views from any Analysis and Design action blocks the Process Action Diagram calls
- Any new views (such as local views)

You map views to fields on screens with Screen Design or Window Design.

During the code generation phase of Construction, you generate a system that consists of screens populated with the fields you initially defined with views. CA Gen bases the generated program logic on the manipulation of views in the Process Action Diagrams.



# Chapter 5: Creating Views

---

This chapter explains the different types of views. You add views to the import, export, local, and entity action view subsets using View Maintenance.

The following table shows which types of views you can add to each view subset.

View Type	View Subset
Entity View	Import, Export, Local, Entity Action
Group View	Import, Export, Local
Work View	Import, Export, Local

Two features, expanding expected effects and Process Synthesis, automatically generate views in the import, export, and entity action view subsets. The features also generate other information including expected effects of processes, definition properties of processes, and process logic.

The following sections describe how to add, detail, and change views.

**More information:**

[Using Process Synthesis](#) (see page 129)

[Building the Action Diagram](#) (see page 11)

## Common Steps for Adding Views

You follow the same steps when you add group, entity, and work views as you do when you add attributes to entity and work views. This section describes these common steps. References to these steps and detailed information about view subsets and view types appear in the sections following the task descriptions.

You can add new attributes or attributes already existing in the model to entity views and work views. You can add attributes in the following cases:

- If you elected not to add attributes when you created a view
- If you discovered additional attributes you want to add to a view after it was created
- If you added a new work set to a work view

Unlike entity views, attributes for a new work set are not available when you first create them because they are not called from an entity type.

## How to Create Views in the Import View Subset

Creating views in an import view subset consists of the following activities:

- Create import group views
- Create import entity views
- Create import work views

### Create Import Group Views

Creating import group views consists of the following activities:

- Add group view
- Add nested repeating group view
- Detail group view

### Add Import Group Views

Adding import group views is the first part of creating import group views. A group view is a collection of one or more associated entity views. Group views have cardinality, that is, each of their component views may occur more than once.

A group view with cardinality greater than one is a repeating group view, and can be thought of as an array of data items.

Group views with cardinality of one are non-repeating group views. Non-repeating group views are often used to collect many other entity views and group views together so that view matching can be simplified when calling subordinate action blocks.

Repeating group views can be illustrated by an order form. A typical order form for one customer relates to attributes from the CUSTOMER, ORDER, PRODUCT, and ORDER LINE entity types in the Data Model. An order relates to only one customer; therefore, an entity view for CUSTOMER exists. The details on the order form concern only one order; therefore, an entity view of ORDER exists.

The views for PRODUCT and ORDER LINE exist because each line on the order represents a different product. However, the correct product for each order line must always be referenced. Therefore, it is convenient to establish a group view that assembles the entity view of PRODUCT and the entity view of ORDER LINE.

The order form can be represented by an import group view named `IN_GROUP_PRODUCT_INFORMATION`. It contains the entity views Input Product (view of `PRODUCT`) and Input Order Line (view of `ORDER LINE`). The group view is defined as repeating because both entity views are imported for each line on the order.

```
group(r) IN_GROUP_PRODUCT_INFORMATION
view of INPUT
entity PRODUCT
attr NUMBER
view of INPUT
entity ORDER_LINE
attr QUANTITY
```

A group view can contain one or more work views. In the next example, the group view `In_Group_Order_Line` contains the work view that includes attributes from the work set `IEF_SUPPLIED`. The group view also contains the input entity view `Input Customer_Order_Line`.

The name of the group view should accurately reflect its contents. Unlike entity view names, group view names always stand alone in the process logic or the procedure logic. They should therefore carry the full weight of the meaning of their components. For example, if you are adding an import group view, you may want to use the naming system shown in the example, which is `IN_GROUP` followed by the name of the entity type.

Procedure Synthesis generates new view names when you specify processes to be implemented by a procedure.

The Consistency Check facility checks group views and subordinate entity views for conformance with the following rules:

- A group view must decompose into subviews.
- An entity view must have at least one predicate view for code generation to succeed.

```
Views for CREATE_ORDER
Import Views
group (r) IN_GROUP_ORDER_LINE
view of CUSTOMER_WORK_AREA
wrk set IEF_SUPPLIED
attr COUNT
attr FLAG
view of INPUT
entity CUSTOMER_ORDER_LINE
attr QUANTITY
attr NUMBER
attr REQUESTING_UNIT
```

**More information:**

[Common Steps for Adding Views](#) (see page 93)

[Using Views](#) (see page 113)

[Specify Import Group View Cardinality](#) (see page 96)

## Add a Nested Repeating Group View

Adding nested repeating group views is the second part of adding import group views.

A group view can include other group views. These subordinate group views are called nested group views. When the nested views have a cardinality greater than one, they are called nested repeating group views.

Nested repeating group views allow flexibility when displaying and performing entity actions on repetitive data, especially attributes of entities involved in one-to-many or many-to-many relationships.

## Detail Import Group Views

The second part of creating group views is to detail them. Detailing group views consists of the following activities:

- Describe group views
- Specify cardinality
- Specify optionality

## Describe Import Group Views

Describing an import group view is an optional activity. It lets you record the contents of a group view when the content is unclear. Keep the description concise.

Click Description to invoke the Description Editor. The Description Editor lets you perform certain word processing functions while you enter the description.

## Specify Import Group View Cardinality

The number of times a group view repeats is referred to as the cardinality of the group view. Group views are the only type of views for which you define cardinality.



When you define a repeating import group view, you specify the following cardinalities:

1. Expected numbers of occurrences for the group view. You set the minimum cardinality of the group view (the least number of occurrences it can contain) by the value you enter for at least. You set the maximum cardinality of the group view (the greatest number of occurrences it can contain) by the value you enter for at most. Finally, you set the average cardinality of the group view (the average number of occurrences it can contain) by the value you enter for on average. To maintain consistency, the values for minimum and average cardinality of repeating group views are adjusted based on the value you enter for the maximum cardinality. For example, if the average cardinality is 12 and you change the maximum cardinality from 20 to 10, the average value is adjusted to 10.
2. Whether the numbers you add for the minimum and maximum cardinalities are estimated or absolute.
3. Explicitly or implicitly indexed. Explicitly indexed repeating group views have user-accessible indexes to refer to specific entries in the repeating group (or table). You must add logic to the Action Diagram to handle explicitly indexed repeating group views.

Implicitly indexed repeating group views allow the subscript of the group view to be incremented implicitly whenever an action occurs that would imply it. For example, if an implicitly indexed repeating group view is referenced in a FOR EACH statement, the index will be incremented whenever the end of the FOR EACH statement is reached.

The Consistency Check facility checks group views for conformance with the following rule:

A group view with cardinality of many must have the expected cardinality defined.

## Specify Import Group View Optionality

The optionality of an import group view refers to whether the view is mandatory (always used as input) or optional (sometimes used as input).

If you specify an import group view as mandatory, each view that belongs to the group view must have a matching view when the action block is invoked.

You can specify optionality only for views you add in the import view subset. Export, local, and entity action views do not have optionality.

## Create Import Entity Views

Creating import entity views consists of the following activities:

- Add import entity views
- Detail import entity views

## Add Import Entity Views

Import entity views are views of entities that are input to a process, procedure, or action block during execution. These views consist of the information input through a screen or window.

The name of an entity view should always modify the entity type name. The combination of entity view name and entity type name must be unique within a process.

In Analysis, if one occurrence of an entity is in the import view subset, you can distinguish import view names from export view names by using a name such as INPUT for entity views. For example, if you add import views for the entities EMPLOYEE, DIVISION, and COST CENTER, the entity views may be named INPUT EMPLOYEE, INPUT DIVISION, and INPUT COST CENTER. This naming convention is useful when you work with long lists of views and want to see the information you are importing quickly.

Based on the needs of information imported to the process, you can add more than one occurrence of an entity to an import view subset. For example, consider the situation where information is required from two separate CUSTOMERS in the same execution of a process. In this case, the process requires two import views of the same entity type, thus making the combination of the entity type (CUSTOMER) name and the view name (input) insufficient to distinguish between the two views. Each entity view must have a unique name.

Procedure Synthesis generates new view names when you specify processes to be implemented by a procedure.

The Consistency Check facility checks entity views for conformance with the following rules:

- An entity view must have at least one predicate view.
- An entity view must have at least one attribute for code generation.

### More information:

[Common Steps for Adding Views](#) (see page 93)

[Using Views](#) (see page 113)

## Detail Import Entity Views

Detailing import entity views consists of the following activities:

- Describe import entity views
- Specify optionality of import entity views

## Describe Import Entity Views

Describing entity views lets you record the contents of an entity view when the content is unclear. As with group views, describing entity views is optional. Keep the description concise and short.

Click Description to invoke the Description Editor. The Description Editor lets you perform certain word processing functions while you enter the description.

## Specify Import Entity View Optionality

All views in the Import view subset must be optional or mandatory. If you specify mandatory, each component attribute view must have a value when the process to which it belongs begins execution.

If you specify optional, a value is not required when execution begins.

You can specify optionality for an optional entity view in the same way you specify optionality conditions for attributes or relationship memberships. You can store this condition as part of the entity view description. The default is optional.

## Create Import Work Views

Creating an import work view consists of the following activities:

- Add import work view (view of work set)
- Detail import work view

A work view defines information that a process, Procedure Step, or action block requires, but cannot be provided from entity types in the model. Work views are applicable when you have an implementation-specific data requirement that you cannot meet by creating a new view of an existing data item. In these instances, you must define a new data item. These data items are not entity types because they do not reflect the reality of the business. However, you may consider them pseudo-entity types because they are composed of attribute definitions.

## Add Import Work Views (Views of Work Sets)

The work view is a view of a work set (also referred to as a work attribute set) that does not create permanent storage. The attributes in a work view come from a work set, not from an entity. Counters, totals, indicators, and selection codes are examples of attributes stored in a work set. You can use these attributes in Action Diagram statements to monitor execution time information such as the calculation of intermediate totals and counts.

Work sets are usually added by the designer, but the software supplies some of them. For example, one work set called IEF\_SUPPLIED is available when you add work views. The attributes in this work set include Select Char, Action Entry, Command, Count, Total Real, Total Currency, Total Integer, Percentage, Average Real, Average Currency, Average Integer, Flag, and Subscript. These views are intended to be used for data that exists only for the duration of an action block or transaction; these views cannot be used directly to update a database.

The following example shows how a work view appears in View Maintenance:

```
Import Views
view of ORDER_PROCESS_COUNT
wrk set IEF_SUPPLIED
attr COUNT
attr TOTAL_INTEGER
attr FLAG
```

Work sets act like entity types in several ways:

- Add a work set to the import, export, or local view subset. Within each view subset, you can add the work view by itself or to an existing group view.
- Place them on screens.
- Pass them through a USE action in the Action Diagram.

Work views cannot be added to the entity action view set, so these views can never be used in entity actions (CREATE, READ, UPDATE, and DELETE).

### Detail Import Work Views

Detailing a work set consists of the following activities:

- Describe a work set
- Add attributes to a work set
- Detail attributes in a work set

### Describe an Import Work Set

Describing the work set of a work view lets you document how the work view is used. As with group and entity views, the description is optional.

Click Description to invoke the Description Editor. The Description Editor lets you perform certain word processing functions while you enter the description.

### Add Attributes to a Work Set

When you start adding work views, you may need to add new attributes to the work sets they reference. This task is described in Common Steps for Adding Views.

## Detail Attributes in a Work Set

Detailing attributes in a work set consists of the following activities:

- Describe attributes in a work set
- Specify properties of work set attributes
- Specify values of work set attributes

The tasks associated with these activities are the same as those in the Data Model.

**Note:** For more information about detailing attributes, see the *Analysis Tools Reference Guide*.

## Create Views in the Export View Subset

You can perform the same types of activities when you create views in the export view subset as when you create views in the import view subset:

- Create export group views
  - Add export group views
  - Detail export group views
  - Describe export group views
  - Specify export group view cardinality
- Create export entity views
  - Add export entity views
  - Detail export entity views
  - Describe export entity views
- Create export work views
  - Add export work views
  - Add export work sets
  - Detail export work sets
  - Detail export work views
  - Describe export work views

These activities require the same steps as those for adding import views, with the following exceptions:

- When you add export views, select the view subset labeled Export Views. Keep this in mind when referring to other sections in this chapter.
- Since the naming conventions for export views and import views are different, use the appropriate naming convention as recommended in this section.
- When you detail a group view, entity view, or work view in the export view subset, you cannot specify optionality. Optionality is available only for import views.

Export entity views are views of entities that are output from a process, Procedure Step, or action block after execution. These views consist of the information displayed on a screen, printed on a report, or passed to another process, Procedure Step, or action block.

The name of an entity view should always modify the entity type name. The combination of entity view name and entity type name must be unique within a process.

If you have one occurrence of an entity in the export view subset in Analysis, you can distinguish export view names by using a name such as OUTPUT. For example, if you add export views for the entities CUSTOMER and ORDER, you can name the export entity views OUTPUT CUSTOMER and OUTPUT ORDER. The name of an export group view might be ORDER\_LINE\_OUTPUT. This naming convention is useful when you want to know at a glance from a long list of views the information you are exporting from a process, Procedure Step, or an action block.

The view subset can contain more than one occurrence of an entity. For example, consider the case where information from two separate CUSTOMERS in the same execution of a process is exported. In this case, the process requires two export views of the same entity type. The combination of the entity type name (CUSTOMER) and view name (output) is insufficient to distinguish between the two views. Each entity view must have a unique name.

Procedure Synthesis generates new view names when you specify processes to be implemented by a procedure.

The Consistency Check facility checks entity views against the following rule:

An entity view must have at least one predicate view.

**More information:**

[Using Views](#) (see page 113)

## Create Views in the Local View Subset

You can perform the same types of activities when you create views in the local view subset as when you create views in the import view subset:

- Create local group views
  - Add local group views
  - Detail local group views
  - Describe local group views
  - Specify cardinality of local group views
- Create local entity views
  - Add local entity views
  - Detail local entity views
  - Describe local entity views
- Create local work views
  - Add local work views
  - Detail local work views
  - Describe local work views

The activities in the previous list require the same steps as those for adding import views, with the following exceptions:

- To define views for the local view subset, use the Action Diagram, Structure Chart, or Action Block Usage tools. The Activity Hierarchy Diagram, Activity Dependency Diagram, Dialog Design, and Screen Design tools do not permit you to create views in the local view subset during view maintenance.
- When you add local views, select the view subset called *Local Views*. Keep this in mind when referring to other sections in this chapter.
- Since the naming conventions for local views and import views are different, name the view using the appropriate convention as recommended in this section.
- When you detail a group view, entity view, or work view in the local view subset, you cannot specify optionality. Optionality is available only for import views.

A local view is a view of an entity type and some or all of its attributes. The sole purpose of a local view is the temporary storage and checking of attribute values.

You use local views primarily in Design. Unlike import and export views, a local view can neither receive data nor present data to screens or calling processes, Procedure Steps, or action blocks except in the context of a USE statement. You cannot use a local view to communicate with the underlying Data Model as you can with entity action views.

As an example of why work views are used, consider the task of locating among all products known to the business, the product with the highest price under \$100. A local view of the entity PRODUCT can be used to achieve this. As each PRODUCT is READ into an entity action view, its price is compared with that in the local view.

To distinguish local views from those views being imported and exported or used as entity action views, you may want to use names such as LOCAL or TEMP. For example, local views for the entity types EMPLOYEE and COST CENTER can be named LOCAL EMPLOYEE and LOCAL COST CENTER. This naming convention is useful when you want to know at a glance from a long list of views the information you are using solely within the process, Procedure Step, or action block.

## Create Views in the Entity Action View Subset

Creating views in the entity action view subset consists of the following activities:

- Add entity action views
- Detail entity action views

Entity action views define stored information that a process, Procedure Step, or action block uses in entity actions (create, read, update, and delete) in the Action Diagram. An entity action view is always an entity view with attributes from a single entity type or subtype. An entity action view cannot be a group view.

## Add Entity Action Views

Use the Action Diagram to add an entity view to the import, export, or entity action view subsets:

1. Specify the entity action view subset as parent view subset for the view
2. Specify the subject entity type or subtype
3. Name the entity view
4. Create any missing attributes for the view
5. Specify attributes

A useful convention, when the entity action view set contains only one view of a particular entity is to leave the entity action view unnamed. This can significantly reduce the wordiness of WHERE clauses in complex READ statements.

The Consistency Check facility checks entity views for conformance with the following rule:

Each entity action view should be used in the action block.



## Detail Entity Action Views

Describing entity action views lets you document how the views are used. Describing entity action views is optional.

Click Description to invoke the Description Editor. The Description Editor lets you perform certain word processing functions while you enter the description.

## Assign View Characteristics

Depending on the view you create, the following characteristics can be assigned:

- Supports entity actions
- Lock required on entry
- Used as both input and output

## Supports Entity Actions

When an entity view is defined in the Import or Export view subset, the Supports Entity Action option specifies whether the view is transient or persistent.

A transient entity view does not support the entity actions CREATE, READ, DELETE, or UPDATE. Transient views can be modified by MOVE and SET actions. Repeating group views always are transient. Transient is the default for entity views in the import, export, or local view subset.

A persistent entity view allows entity actions to be performed on the designated view. All rules that apply to entity action views apply to persistent entity views. Persistent views can be thought of as database buffers.

Use persistent views when multiple Action Diagrams require access to the same physical data record, as in batch processing.

Support for entity actions reduces the need for multiple READs of the same entity action views within a process. After the entity action view is read, a persistent view may be passed to subordinate action blocks. This means that a Procedure Step can perform a READ and pass the persistent view to an action block that performs an UPDATE action. A lock on the database record is released at the end of the Procedure Step. The subordinate action blocks may perform an UPDATE action on a received persistent view.

The following example shows persistent views in a Procedure Step and an action block:

Procedure Step:

```
MAINTAIN EMPLOYEE
IMPORTS:...
EXPORTS:...
ENTITY ACTIONS:
Entity View employee
name
number
CASE OF COMMAND
CASE read
USE AB_READ_EMPLOYEE
WHICH IMPORTS...
WHICH EXPORTS output_employee employee
CASE update
READ employee
USE AB_UPDATE_EMPLOYEE
WHICH IMPORTS employee
.
.
Action block:
AB_UPDATE_EMPLOYEE
WHICH IMPORTS employee
.
.
Action block:
AB_UPDATE_EMPLOYEE
IMPORTS:
Entity View input_persistent employee (mandatory, persistent, locked
.
.
IF input_persistent employee IS POPULATED
UPDATE input_persistent employee
```

Consider this example to understand the benefit of persistent views. An Action Diagram READ\_EMPLOYEE performs an initial READ of an entity occurrence. The entity action view in the initial Action Diagram is matched to an import entity view in a USED action block where an entity action is performed. The receiving import entity view is designated persistent, exported, locked.

#### **Persistent**

Indicates entity actions can be performed on the view.

#### **Export**

Indicates the import entity view can be modified. Data can flow out of the action block without the need to explicitly move the data to an export view.

**Lock**

Indicates the view does not need to be READ again.

To be used for entity action statements, import views must be both persistent and used as both input and output. Then, import views can be used in the following cases:

- In statements that include the verbs READ, READ EACH, UPDATE, CREATE, DELETE, TRANSFER, ASSOCIATE, and DISASSOCIATE
- Within the qualifiers of READ and READ EACH statements
- As the target of predicate action set verbs

Only persistent entity views and entity action views can be used in IF statements that test whether the view is populated or locked. For more information about locked occurrences, see the next section.

**Downstream Effects:** There is a performance advantage to be gained by using persistent views that are exact matches. Two view structures are exact matches when they contain the same entity view and the same attributes in the same order.

If an entity action view or a persistent view is matched exactly to a persistent view in a subsequent action block, then all references between the two are handled by address passing. View data is not moved to the called action block.

Matching views exactly is quicker than actually moving the data and having two copies of it. Working storage is built in the calling Action Diagram for the view and Linkage section (only) copies of the "database buffer" are built into the called action block.

The exact match performance consideration also applies to transient views. Exported on Import Views and Imported on Export Views eliminate the overhead of extra views. SETs and MOVEs to attributes in the import view are allowed, and export views are not initialized upon entry.

## Lock Required on Entry

The Lock Required on Entry option allows locking of an occurrence in an import view at entry to the action block. Locking an occurrence prevents other transactions from changing the view. When you add an entity type to a view, this characteristic becomes available for selection only if the view supports entity actions.

Those views in which occurrences are to be locked are labeled locked in the Action Diagram view list. Those views in which occurrences are not locked are labeled unlocked. The default value is unlocked.

The Locked parameter is strictly a means of communication at code generation time. The generator checks the view matching on USE statements to determine the intent of called action blocks regarding the views that are being passed. Use the Locked parameter on the receiving persistent view if any entity actions are planned.

**Downstream Effects:** When an action block or procedure reads an entity action view that is matched to a view in a called action block marked 'locked,' this signals code generation to create a cursor with locking. The implication is that the called action block intends to update the database using this matched entity action view.

## Used as Both Input and Output

The Used as Both Input and Output option allows import entity views to be exported and export entity views to be imported. Import views that are to be used for both input and output are labeled exported on the Action Diagram view list. Import views that are to be used for input only are labeled import only. Export views that are to be used for both input and output are marked imported; export views that are to be used for export only are labeled export only.

This characteristic cannot be set on the import and export view sets of Procedure Steps; it can be used only in action blocks.

Using a view as both input and output lets a single view act as both import and export views. This facilitates transaction restart or availability of information on a return from a link.

The following possibilities can occur when import views are used as both input and output:

- SET and MOVE may be used to change data.
- The import view can be changed and the changes can be returned to the calling action block.
- Import views used for export do not need to be moved to export at the start of each Action Diagram.

Import views that are *not* used for both input and output may not be matched to views that are used for both input and output; the calling view cannot be changed on return from the called action block.

For export views, the view is not initialized on entry to the action block. The export view may also receive data into action blocks. When export views are selected for import, any values in the view of the calling action block pass to the called action block.

**Downstream Effects:** Import views selected to be used as both input and output are updated in the called action block. The calling action block also sees the results. For export views selected to be used as both input and output, the called action block sees data passed from the calling action block.

Using a view as both input and output allows values to be returned to entity views using call by reference instead of call by value.

## Recommendations

Consider the following tips as you create views:

- When passing data up and down a hierarchy of Action Diagrams, use one of the following combinations of view characteristics:
  - Import, Transient, Exported
  - Export, Transient, Imported
- When importing data that will not be exported, use Import, Transient, Import only. These views can be starved.
- When multiple Action Diagrams require access to the same physical data record, use persistent views. Batch processing is an example of multiple Action Diagrams requiring access to the same physical data record.
- Remember that persistent views require coordination between the using and the used Action Diagrams as to the intended usage of the data. It is not a good idea, for example, to lock entity occurrences that are needed only for inquiry. A view can only be Locked or Unlocked. There is no option to switch between the two options at runtime.
- Starve all views, especially entity action views. Include only the attributes that are required to make an Action Diagram work. Starving the views this way saves time at execution because the DBMS does not retrieve any unnecessary data.

## How to Change Views

As you work with views in the Analysis and Design tools, you can change the various components, change their placement, or change information that the views reflect.

The following list contains changes you can make while working with views.

- Change parent of views
- Copy views
- Delete views and their attributes
- Make miscellaneous view changes

- Move a view and reorder a view under a parent
- Rename a view

## Change Parent of Views

The parent of a view (entity view, work view, or group view) is either a group view or the view subset (import, export, or local views). If a view is part of a group view, its parent is the group view. If a view is not part of a group view, its parent is the view subset. You can change the parent of a view from one group view to another group view, a group view to a view subset and a view subset to a group view.

**Note:** When you change the parent of a view, both the old parent and the new parent must be within the same view subset.

## Copy Views

You can copy views as described in the following table. In all cases, attributes are copied as part of the view to which they belong.

Copy from	Copy to	Cannot copy	Tool Required
Entity views Subtype views Work views	Export view subset	Group views Predicate views	Unrestricted
Entity views Subtype views Work views	Import view subset	Group views Predicate views	Unrestricted
Entity views Subtype views Work views	Local view subset	Group views Predicate views	Action Diagram
Entity views Subtype views	Entity action view subset	Group views Work views Predicate views	Action Diagram

## Delete Views

A view can be deleted if it is not referenced in an Action Diagram. When a group view is deleted, the generated application software also removes all entity views associated with the group view and all of the predicate views within the entity views.

## Move Views

You can move a view to another position under a parent. You cannot move views between the import, export, and entity action view subsets, but can move a view to and from the local view subset. You cannot move an attribute view to a different parent.

**Note:** Do not move attribute views in a repeating group view if a FILTER or SORT statement is used. The FILTER and SORT statements refer to an attribute view according to its sequence in the repeating group view. By moving the views, you can change the sequence of attribute views and unknowingly change the arguments in a sort or filter statement.

## Rename a View

You can rename an entity, group, or work view to better reflect the business environment.





# Chapter 6: Using Views

---

This chapter explains how to use views. After creating views, you can use them in the following ways:

- Associate views of processes with external objects in the Activity Dependency Diagram
- Synthesize views in the Dialog Design
- Match views

## Associate Views with External Objects (Add)

When you join an activity and an external object in the Activity Dependency Diagram, you identify the information to be passed between them. To do this, you assign associated views that are subsets of the import or export views of the activity.

You identify to a process what data to expect from the external object and what data to prepare to send out. This level of detail is recommended when two or more external objects provide information to or receive information from the process. In these cases, the associated views help identify the providers and receivers of information.

You can associate only import and export entity views of a process with an external object. You cannot associate group and work views. The local and entity action view subsets are not available in the Activity Dependency Diagram.

## Synthesize Views in the DLG

When you conduct procedure synthesis in the Dialog Design, the following activities occur automatically:

- Synthesizes the views for each process implemented in the procedure
- Adds a CASE OF COMMAND statement with nested USE statements that call the implemented action blocks
- Matches the views for the action blocks

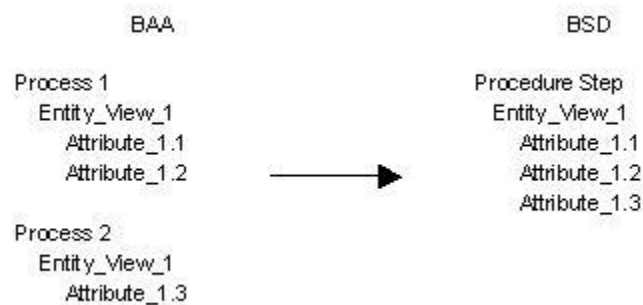
During procedure synthesis, the software generates a merged set of views for the Action Diagram of the procedure based on the combined views of all elementary processes you chose to implement in the procedure. The software synthesizes views based on the type of process-to-procedure implementation you invoke:

- One-to-one
- One-to-many

- Many-to-one
- Many-to-many

The most common implementation is one-to-one. Use one-to-many for alternative implementations of the same elementary process. Use many-to-one when multiple elementary processes have similar views that a common set of users require. Many-to-many is a combination of the one-to-many and many-to-one cases. The following illustration shows how views are synthesized when you invoke a many-to-one process-to-procedure implementation.

The following illustration is a View Synthesis for Many-to-One Process-to-Procedure Implementation:



During procedure synthesis, the software copies the views and assigns new view names. The following table shows the default names assigned to entity and group views during view synthesis. The table includes names for additional view occurrences. This table illustrates the Default Names for Synthesized Views:

View Type	Import View Subset	Export View Subset
Entity View	IMPORT	EXPORT
EV-2nd Occurrence	IMPORT_2	EXPORT_2
Group view	GROUP_IMPORT	GROUP_EXPORT
GV-2nd Occurrence	GROUP_IMPORT_2	GROUP_EXPORT_2

You can rename views and add names to the entity action views by accessing view maintenance in the Activity Hierarchy Diagram, Activity Dependency Diagram, or Action Diagram.

**Note:** Procedure synthesis leaves unchanged any default name you assign prior to transformation.

When you specify that a process containing two or more views of the same entity be implemented, the software renames the views with sequential numbers. The following table shows examples of renamed import and export entity view occurrences of EMPLOYEE. It also shows examples of renamed import and export group view occurrences of ORDER\_LINE.

This table illustrates the View Naming examples:

Analysis	Design
Input_Employee New_Employee	Import_Employee Import_2 Employee
Output_Employee Old_Employee	Import_Employee Import_2 Employee
In-Group_Order_Line Input_Group_Order Line	Import_Employee Import_2 Employee
Out_Group_Order_Line Output_Group_Order_Line	Import_Employee Import_2 Employee

**More information:**

[Procedure Synthesis](#) (see page 157)

[Using Procedure Synthesis](#) (see page 159)

## Match Views

View matching is how the software defines information passed between an Action Diagram and an action block. You can match views alone or match and copy views. In Analysis, you can match views with the Action Diagram, Structure Chart, and Action Block Usage tools. In Design, you can match views with the Action Diagram, Dialog Design, Structure Chart, and Action Block Usage tools.

You can also unmatch views you previously matched.

**Note:** For more information about matching views using the Dialog Design tool, see the *Design Guide*.

## Guidelines for Matching Views

The following view matching guidelines are enforced.

- Match views between two entity views of the same entity type. This guideline also applies to work views.
- Match views between two group views with comparable structures. The two group views have comparable structures in the following cases:
  - The entity views and group views that make up each group view to be matched correspond exactly in content and relative position.
  - For repeating group views, the maximum cardinality of the receiving group view is greater than the maximum cardinality of the sending group view.
  - For each component group view matched to its corresponding group view, the two group views look exactly alike except for the list of attribute views they contain.
- Match views between two group views when the cardinalities are compatible. That is, the receiving maximum cardinality of the view must be greater than or equal to the sending maximum cardinality of the view; otherwise, information is lost.

**Note:** Passing repeating group views can present a tremendous overhead compared with passing a single occurrence. The overhead can be reduced by exactly matching the attributes, cardinality, and the order of entity and work groups between the sending and receiving views.
- In USE and SET USING statements in the Action Diagram, match a single occurrence of a repeating group view to a non-repeating group view of comparable structure.
- When you match dialog flows, match export views of the source Procedure Step to import views of the target Procedure Step.
- When you match views in Action Diagrams:
  - Match import views of the used action block to the import, local, or entity action views of the action block that USEs.
  - Match export views of the USEd action block to the export or local views of the action block that USEs, or to export views marked as importable. Also, import views in the calling action block marked as exportable can be matched to USEd action block export views.

View matching causes the values of attribute views in the source view to be copied to the corresponding attribute views in the target view. Attribute views correspond when they implement the same attribute.

The table, View Matching Possibilities by Characteristic, indicates which views can be matched based on view characteristics.

The following diagram shows the View Matching Possibilities by Characteristic:

			Called action block							
			Import view				Export view			
			Transient Import	Export	Persistent Import	Export	Transient	Persistent		
Import view	Calling action block		Import	X						
	Transient	Export		X	X			X		
	Persistent	Import	X		X					
		Export	X		X	X			X	
Export view	Transient	Import	X	X				X		
	Persistent	Import	X		X	X			X	
	Entity/action views	Export	X		X	X			X	
		Local views	X	X				X		

The Consistency Check facility checks matched views for conformance with the following rules:

- Matched entity views should have at least one attribute in common.
- Matched group views should have the same entity view structure.
- Matched views must have the same cardinality.
- No view matches are allowed under a repeating group view.

**More information:**

[Creating Views](#) (see page 93)

## Match Views Using the PAD

Matching views using the Action Diagram tool consists of the following activities:

- Add USE control actions
- Update USE control actions
- Add SET USING assignment actions

A process, Procedure Step, or action block can use the logic of another action block. The information passed to and returned from the used action block is defined by view matching in the Action Diagram of the process or Procedure Step that uses the action block.

## Add a USE Control Action

Adding a USE control action consists of the following activities:

- Specify USE control action
- Create new action block
  - Match called action block import views
  - Match called action block export views
- Use existing action block
  - Match called action block import views
  - Match called action block export views

Adding a USE action in the Action Diagram is one way to access view matching in the PAD. The other two ways are discussed in the following sections. You can match views after you add the USE action and either create a new action block to use or specify an existing one.

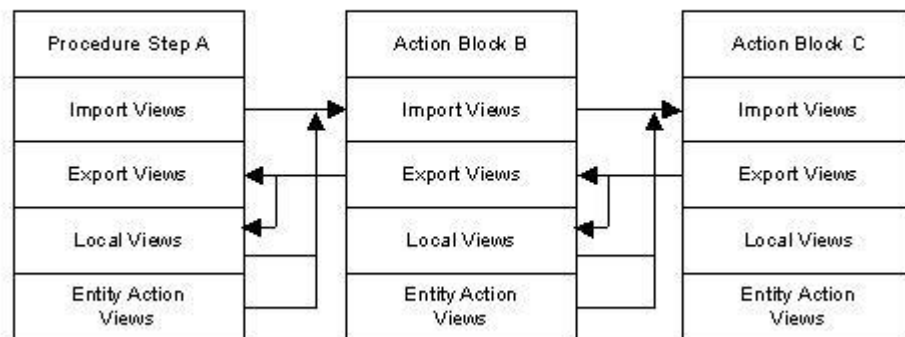
A process, Procedure Step, or action block calls other action blocks using the USE control action. When you invoke a USE action, you need to match the views of the called action block to the views of the calling Procedure Step or action block. The matching occurs in the Action Diagram of the calling (current) Procedure Step or action block.

When you match import views, the import views of the called action block are listed with possible supplying views of the calling process, Procedure Step, or action block. The supplying views may be import views, entity action views, or local views. Work attributes cannot be matched to entity type attributes.

The following illustration summarizes import view matching:

Called Action Block		Calling Process Procedure Step or Action Block
Import Views	MATCHED TO:	Supplying Views Import Views Local Views Entity Action Views

#### View Matching Using Action Diagramming



When you have matched the import and export views of the action block, the USE statement is complete. Matched views appear in the Action Diagram as shown in the following examples of USE statements:

```
USE find_space
WHICH IMPORTS: Group View product_information
WHICH EXPORTS: Entity View new_warehouse
USE dsgn_create_order
WHICH IMPORTS: Group View group_import
WHICH EXPORTS: Group View group_export
```

### Update a USE Control Action

Updating a USE control action is the second way to access view matching in the Action Diagram. Updating a USE control action consists of the following activities:

- Specify view in USE control action to change
- Match called action block import views
- Match called action block export views

## Add a SET USING Assignment Action

Adding a SET USING action consists of the following activities:

- Specify SET action
- Specify USING action
- Match called action block import views

Adding a SET USING action is the third way to access view matching in the Action Diagram. The SET USING action lets you use an algorithm to set an attribute value. The algorithm is defined as an action block. When you invoke a SET USING action, you need to match the import views of the called (USED) action block to the supplying views from the calling Procedure Step or action block. The export view of the called action block is the value of the attribute you are setting. When you have matched the import views of the action block, the SET USING statement is complete.

The following example shows how a SET USING statement appears in the Action Diagram:

```
SET status USING status_assignment  
WHICH IMPORTS: Entity View new_employee
```

## Match and Copy Views

You use the match-and-copy function for import and export view matching in the Action Diagram when the calling process (or action block) does not have a supplying or receiving view that corresponds to the view in the called action block. The software copies the view from the used action block to the calling process or action block and then performs view matching.

## Unmatching Views

You can also unmatch import and export views using the Action Diagram tool. Unmatching a view *does not* delete either the supplying or receiving view. If you use the match-and-copy function to create a view and then you unmatch the views, the created view is not deleted.



## Maximum Size of Views

The following restrictions apply to the maximum size of views:

- For online-based applications, the maximum size of views is 32 KB for import and 32 KB for export.
- For a single server load module, the maximum size of views is 32 KB for import, and 16.7 MB for import and export.
- For GUI applications developed in Windows, the maximum size of views is unlimited for import and export.

The amount of data that can be passed on a dialog flow depends on the type of application, the load module packaging, and the execution environment. The following restrictions apply for the view sizes:

- For Procedure Steps packaged in an online Load Module, the maximum view size that can be passed from one load module to another load module is 31,744 bytes.
- For Windows GUI applications, the maximum view size that can be passed from one Procedure Step to another Procedure Step within the same load module or another load module on the same machine (that is, not a client-server flow) is unlimited.
- For Procedure Steps packaged in a cooperative load module, the maximum view size can vary from 31400 bytes to 16777215 bytes. The maximum varies depending on the server platform and communication protocol used. View sizes less than 31,400 are supported and are the most portable. To determine the limits that are currently supported for each server/communications combination use Consistency Check.



# Chapter 7: Process Synthesis

---

This chapter discusses Process Synthesis, a process logic-building feature accessible from more than one diagram. CA Gen lets you build process logic in Action Diagrams by performing the following tasks:

- Building from scratch
- Expanding expected effects
- Using Process Synthesis

These methods require varying degrees of effort and are suitable for different situations.

- Building process logic from scratch requires you to select each command from a menu. This method may take more time to complete and assumes familiarity with the Action Diagram tool, but it offers great flexibility and control.
- Expanding expected effects is an interactive method that recalls the effects you predicted for each process and guides you through the options available to build action statements. This method requires less familiarity with the Action Diagram tool. However, after you have defined logic with this method or any other method, you must still use individual commands to specify additional logic.
- Process Synthesis allows you to generate processes and detailed process logic automatically based on a consistent portion of the Data Model. The generated processes and process logic reflect the five basic entity actions available in Process Synthesis—CREATE, READ, UPDATE, DELETE, and LIST (READ EACH).

This chapter discusses the following concepts:

- When to use Process Synthesis
- What is generated by Process Synthesis

This chapter also contains examples of generated output from Process Synthesis.

## When to Use Process Synthesis

You may find that using Process Synthesis is faster and easier than building Action Diagrams from scratch or expanding expected effects. Process Synthesis looks at and uses entity types related to the subject entity type (the entity type upon which a generated process operates), creating the required views and action statements. Using related entity types (entity types with a direct relationship to the subject entity type) is especially helpful if the entity type being acted upon has multiple relationships.

Process Synthesis makes certain assumptions about your intentions based on the Data Model. In comparison, expanding expected effects makes no assumptions and offers you the possible choices. Sometimes you may prefer to expand expected effects because assumptions made by Process Synthesis may not be what you want.

## Process Synthesis Output

The generated output from Process Synthesis does not differ from the types of output you obtain when you build from scratch or expand expected effects. For each elementary process, Process Synthesis creates the following information in various diagrams:

- Elementary process box in the Activity Hierarchy Diagram
- Views for the elementary process
- Expected effects for the elementary process
- Definition properties for the elementary process
- Process logic in the Action Diagram
- Complete logic except for specific action statements needed to reflect your business rules

**Note:** For the best and most complete results, resolve all consistency check error messages pertaining to the Data Model before invoking Process Synthesis.

The five types of entity actions that you can generate through Process Synthesis are described next:

### CREATE

Creates an occurrence of the specified subject entity type. Also may create or read related entity types as specified in the dialogs that appear.

### READ

Retrieves an occurrence of the subject entity type and places its contents in an entity action view. Also may read related entity types, if identifying relationships are present.

### UPDATE

Reads and then modifies an occurrence of the subject entity type. Also may read, associate, disassociate, or transfer related entity types as specified in the dialogs that appear. CA Gen generates all ASSOCIATE actions for any mandatory relationship when a CREATE action is added to the Action Diagram.

**DELETE**

Reads and then removes occurrences of the subject entity type from the database.  
Also may read related entity types as specified in the dialogs that appear.

**LIST**

Reads each occurrence of the subject entity type.

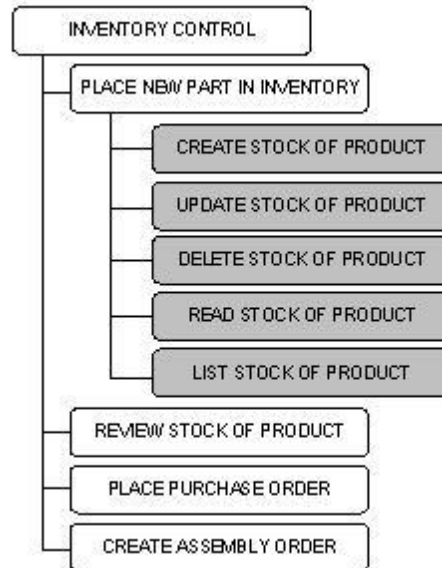
The following example shows the results of Process Synthesis. The process Place New Part In Inventory could be expected to create, read, update, and delete entities of the type Stock of Product.

The following example lists the candidate processes generated in the Activity Hierarchy Diagram to perform the indicated activities, as well as the list activity (READ EACH).

**Names of Generated Processes**

CREATE\_STOCK\_OF\_PRODUCT  
READ\_STOCK\_OF\_PRODUCT  
UPDATE\_STOCK\_OF\_PRODUCT  
DELETE\_STOCK\_OF\_PRODUCT  
LIST\_STOCK\_OF\_PRODUCT

The following illustration shows the new processes generated by Process Synthesis:



The following example shows part of the process logic and the views generated for Create Stock of Product. This code was generated at the same time the processes were created in the Activity Hierarchy. Process Synthesis also populates appropriate cells in the Elementary Process or the Entity Type Matrix.

**Note:** For more information about matrices, see the *Analysis Tools Reference Guide*.

### Generated Process Logic

```
CREATE_STOCK_OF_PRODUCT
IMPORTS: ...
EXPORTS: ...
LOCALS:
ENTITY ACTIONS: ...
READ warehouse
WHERE DESIRED warehouse name IS EQUAL TO input warehouse
name
WHEN successful
MOVE warehouse TO output warehouse
READ product
WHERE DESIRED product number IS EQUAL TO input product
number
WHEN successful
MOVE product TO output product
CREATE stock_of_product
ASSOCIATE WITH product WHICH is_inventoried_as IT
ASSOCIATE WITH warehouse WHICH holds IT
SET quantity_on_hand TO input stock_of_product
quantity_on_hand
SET color TO input stock_of_product color
WHEN successful
MOVE stock_of_product TO output stock_of_product
WHEN already exists
WHEN not found
```

## Prerequisites

Before you can access the commands to invoke Process Synthesis, you must invoke the CA Gen software and access an existing model or create a new one.

You can perform Process Synthesis in a model in which the data portion (in the Data Model) is consistent and stable. You can generate processes in an inconsistent model but the resulting Action Diagram may not be as complete as required.

During Process Synthesis you can perform or bypass the consistency check. If you perform the check and it produces no errors or only warnings, Process Synthesis can execute successfully. If consistency check issues error messages, Process Synthesis will not let you proceed until you correct all the errors. If you elect to bypass the check, Process Synthesis proceeds without a consistency check.

**Downstream Effects:** Processes generated during Analysis are the basis of procedures used in Design. The stability of the Data Model becomes important during Procedure Synthesis, which you conduct in Design.

For example, if you change the identifier of an entity type used by a generated process, the implementation of processes as procedures will fail a consistency check. You must then correct the problem. You may need to delete the process (in Analysis) and recreate it before you can successfully implement it in Design.

If you have added logic to the Action Diagram, you may decide to correct the specific error condition instead of deleting and regenerating the process. When you delete a process, you delete all the information stored for it, including its views, expected effects, and action statements. To avoid these losses and unexpected results, always perform a consistency check and correct any errors before you attempt to implement processes into procedures.

**Note:** If you plan to perform Process Synthesis on a subset of a model, ensure to scope the entire Data Model for inclusion in the subset. If you fail to do this, the Action Diagrams you generate may be incomplete because you do not have access to entity types directly or indirectly related to the subject entity type.

## How to Access Process Synthesis

You can access Process Synthesis directly from the following diagrams:

- Data Model
- Data Model List
- Data Model Browser
- Activity Hierarchy Diagram
- Activity Dependency Diagram
- Action Diagram

## Process Synthesis from the Data Model

In the Data Model, available processes can be defined as elementary or not elementary. Elementary processes appear on a pop-up for selection only if they do not have actions defined for them in Action Diagrams. Non-elementary processes appear on a pop-up for selection only if they do not have subordinate processes. If you select a non-elementary process, it changes to elementary as a result of Process Synthesis.

## Process Synthesis from the Action Diagram

In the Action Diagram, you can expand expected effects only if you have selected an elementary process for which you have defined expected effects in the Activity Hierarchy Diagram or Activity Dependency Diagram.

You can access Process Synthesis by accessing an Action Diagram that has no statements from the following tools:

- Structure Chart
- Action Block Usage
- Dialog Flow Diagram
- Screen Design
- Prototype

**Note:** You can access Process Synthesis from existing processes, Action Diagrams, and action blocks only if they do not contain action statements. They may have defined views. All predefined views are deleted at the start of Process Synthesis.

## Process Synthesis from Activity Hierarchy and Activity Dependency

In the Activity Hierarchy Diagram and Activity Dependency Diagram tools, Process Synthesis lets you generate processes and process logic for all five basic entity actions at the same time. You usually use Process Synthesis when building Action Diagrams for basic processes.



# Chapter 8: Using Process Synthesis

---

Using Process Synthesis consists of the following three main activities:

- Creating a process logic diagram
- Performing Process Synthesis
- Reviewing results of Process Synthesis

Each of these activities has subordinate activities.

Although Process Synthesis lets you generate logic while you are building your data and activity models, you can wait until you are ready to perform interaction analysis with the Action Diagram tool before you use Process Synthesis.

The following sections contain examples of Process Action Diagrams generated for create, read, update, delete, and list actions. The explanations in these sections assume you have an understanding of data modeling objects, process logic, and Action Diagram keywords and statements.

**Note:** For more information about data modeling, see the *Analysis Tools Reference Guide*. For more information about process logic and Action Diagram keywords, see the chapter "Building the Action Diagram."

**More information:**

[Building the Action Diagram](#) (see page 11)

## How to Create a Process Logic Diagram

One approach you can use to address process logic analysis is to create a process logic diagram. This is the first step before using Process Synthesis and consists of the following activities:

- Select elementary process to analyze
- Identify entity types used in the process
- Identify required actions
- Identify sequence of actions

Creating a process logic diagram is a manual activity. A specific CA Gen tool is not available for this function. It is fundamental to understanding both the specific effects of Process Synthesis and process logic in general.

The process logic diagram helps you understand the logic of a specific process before detailing it precisely in the Action Diagram. Look at the main (subject) entity types that the process affects and then consider whether any additional, related entity types are affected. After you have determined which entity types are necessary in your process logic, specify the different entity actions against each entity type and attempt to sequence them. Record this information in your process logic diagram.

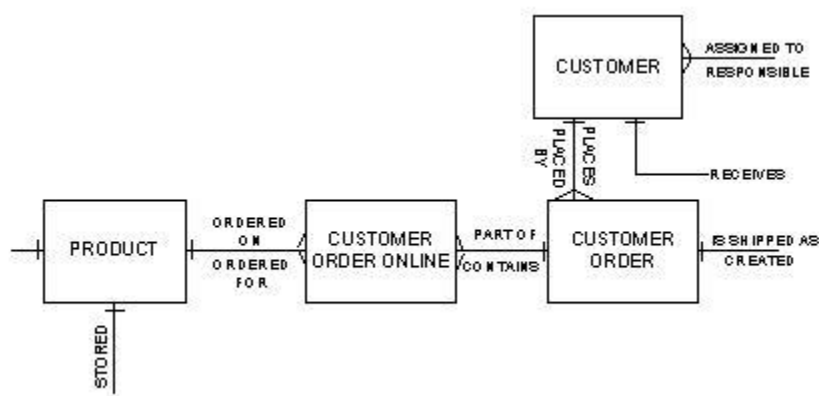
## Select an Elementary Process to Analyze

Selecting an elementary process to analyze is the first step in creating a process logic diagram. Start by analyzing the Data Model to see what activities can be implied by its entity types. For example, the portion of the Data Model shown in the Take Order Process Example contains the entity types CUSTOMER ORDER, CUSTOMER ORDER LINE, and CUSTOMER. For the business rules demonstrated by the relationships, if CUSTOMER places an order, we must create an occurrence of ORDER. You can create and select an elementary process called Create Customer Order for the subject of your analysis.

At this point, you should also consider the following details for the process:

- Definition
- Import and export views
- Identification as an elementary process

The following example shows the Take Order Process:



## Identify Entity Types Used in Process

This step involves identifying the primary entity types, which means looking first at the subject entity type. For example, if the process name is Create Customer Order, the subject entity type is a CUSTOMER ORDER.

The second part of identifying entity types is to study the neighborhood of the primary entity types. The neighborhood is the set of all entity types directly related to the subject entity type. Based on the entity action to be performed against the subject entity type, determine if any related entity types should be included in the logic for a particular process. Based on the relationships involved, you may decide to include a related entity type in the process logic. You then need to look at the neighborhood of the related entity type and determine if any additional entity types are required.

In the Take Order Process example, CUSTOMER ORDER is related to CUSTOMER ORDER LINE because they are joined in the Data Model. Because CUSTOMER ORDER LINE cannot exist without one PRODUCT (denoted by the mandatory relationship), PRODUCT has an implicit relationship to CUSTOMER ORDER.

## Identify Required Actions

In this step, you annotate the Data Model with the five entity actions (CREATE, READ, UPDATE, DELETE, and LIST (READ EACH)). The entity actions are performed on occurrences of the identified entity types.

The logic for the Create Customer Order process should reflect the following actions:

- CREATE CUSTOMER ORDER
- READ CUSTOMER
- READ PRODUCT and ASSOCIATE
- CREATE CUSTOMER ORDER LINE

You also determine which actions on attributes and relationships of the affected entity types are used and how. This gives you an idea of the three types of actions needed in the process logic:

- Required SET actions (1) that assign values to attributes in CREATE and UPDATE action statements. A CREATE action must SET each mandatory attribute of the subject entity type. In addition, it can SET optional attributes as required by the process.
- Required ASSOCIATE (2) and DISASSOCIATE (3) actions that establish and eliminate pairings in the CREATE action statements.

## Identify Sequence of Actions

After identifying all actions on entity types, attributes, and relationships, you can determine the order in which those actions should be performed during process execution.

For example, in the Create Customer Order process, you can perform the following actions:

1. READ customer.
2. CREATE order and ASSOCIATE WITH customer.
3. READ product.
4. CREATE order line and ASSOCIATE WITH product and order.

Before pairing two occurrences of entity types, you must make them available through a READ or CREATE statement. In general, you should read existing entities, then create entities with which they will be associated.

When the process logic diagram is complete, you have a picture outlining the process logic. You can use the results of this activity with Process Synthesis to generate the Process Action Diagram for a process. The Process Action Diagram logic contains all the required entity actions.

## Perform Process Synthesis

You can perform Process Synthesis from any of the following diagrams:

- Data Model, Data Model Browser, or Data Model List
- Activity Hierarchy Diagram and Activity Dependency Diagram
- Action Diagram

The following table shows the output from each of the tools from which you can initiate Process Synthesis. The following table also illustrates the Process Synthesis Tools and Output:

Tool	Output
Activity Hierarchy Diagram Activity Dependency Diagram	Elementary process box
Data Model Data Model List Data Model Browser Activity Hierarchy Diagram Activity Dependency Diagram	Definition properties
Data Model Data Model List Data Model Browser Activity Hierarchy Diagram Activity Dependency Diagram	Expected effects

Tool	Output
Data Model Data Model List Data Model Browser Activity Hierarchy Diagram Activity Dependency Diagram Action Diagram	Views
Data Model Data Model List Data Model Browser Activity Hierarchy Diagram Activity Dependency Diagram Action Diagram	Action Statements  <b>Note:</b> The generated logic is complete except for specific action statements needed to reflect your business rules.

The requirements to perform Process Synthesis differ from tool to tool. These requirements are covered in the following sections.

## Process Synthesis Rules

This section describes the rules used to populate the default values for CREATE, UPDATE, and DELETE statements generated during the Process Synthesis dialogs. Each dialog allows different selections and defaults that are entirely dependent upon the entity action type and the relationships involved. In the following rules, the relationships are defined as they appear in the Data Model, with the subject entity types always to the left of all possible related entity types. The rules apply to creating, updating, or deleting a single occurrence of the subject entity type. The following illustration shows the Representation of Relationships in Process Synthesis Rules:



In general, a mandatory one-optional many relationship means the subject entity type sometimes has one or more related entity types and the related entity type always has one subject entity type. The source relationship refers to the mandatory one and the destination relationship refers to the optional many.

The following relationship combinations are grouped by the actions allowed against the target entity types.

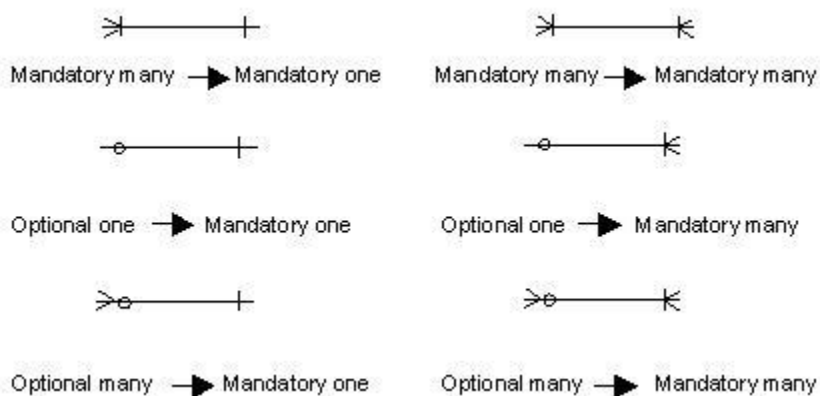
## Rules for CREATE

Use these combinations of relationship memberships to determine whether you can use CREATE and READ together.

- For these relationship combinations, the default is CREATE - yes, and READ is not allowed. CREATE - Yes, READ not allowed:



- For these relationship combinations, the default is CREATE - no, and READ is allowed. CREATE - No, READ is allowed:



- For these relationship combinations, the default is CREATE - no, and READ is not allowed. CREATE - No, READ not allowed:

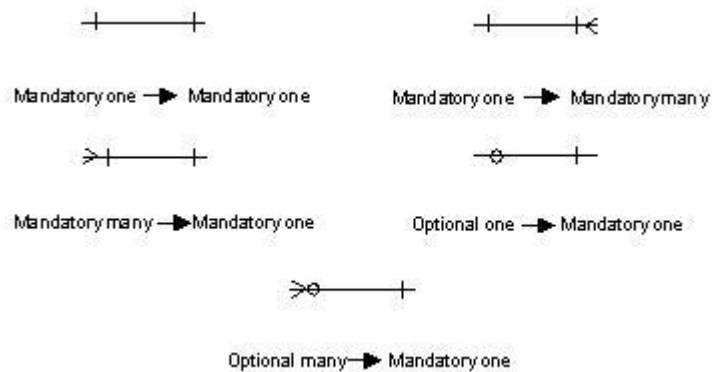


**Note:** All relationships (any one or any many) for which you indicate a CREATE action cause a FOR EACH construct and the necessary repeating group views to be created.

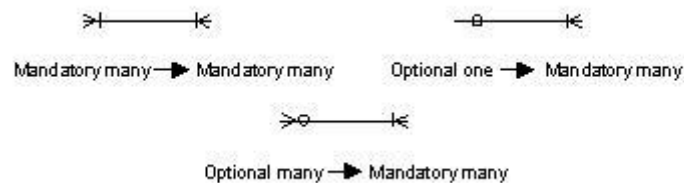
## Rules for UPDATE

Use these combinations of relationship memberships to determine whether you can use TRANSFER, ASSOCIATE, and DISASSOCIATE actions together.

- For the following relationship combinations, only TRANSFER actions are allowed (ASSOCIATE and DISASSOCIATE are not allowed):



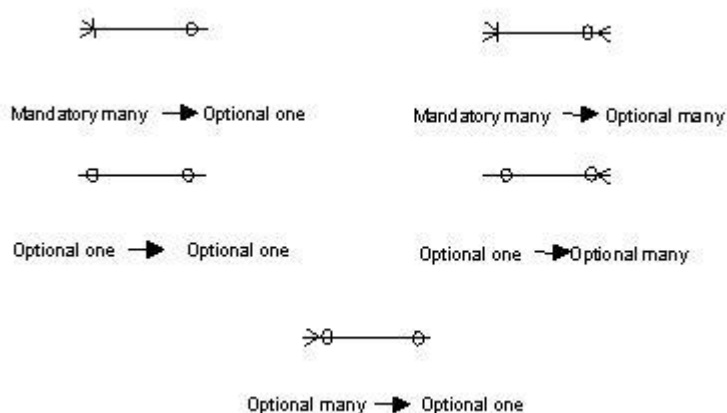
- For the following relationship combinations, the default is ASSOCIATE - NO. DISASSOCIATE is not allowed, and TRANSFER actions are allowed:



- For these relationship combinations, the default for DISASSOCIATE - NO, TRANSFER actions *are* allowed, and ASSOCIATE is *not* allowed:



- For the following relationship combinations, the default for both ASSOCIATE and DISASSOCIATE - NO, TRANSFER actions *are* allowed:



**Note:** Associate actions require that an occurrence of the target entity type already exists. Therefore, the identifier is included in the READ statement before the ASSOCIATE statement. However, DISASSOCIATE and TRANSFER actions require that an occurrence of the subject entity type is already associated. The READ statements for these actions are qualified using related to subject entity type instead of using identifiers. For DISASSOCIATE statements, a READ EACH statement is built if the relationship is any any-many.

TRANSFER actions are only allowed whenever the property of the relationship between the subject entity type and the related entity type has been defined as transferable.

## Rules for DELETE

If you want to delete a single occurrence of the entity type you originally selected, all related entity types are allowed to be read.

**Default:** No

## Process Synthesis from Data Model

When you initiate Process Synthesis from the Data Model, you generate action statements for one process at a time. The process must already exist in the Process Hierarchy or Process Dependency Diagrams. Each process may be defined as elementary or not elementary in its definition properties. Elementary processes appear only if they do not have actions defined for them in Action Diagrams. Non-elementary processes appear only if they do not have subordinate processes (children). If you select a non-elementary process, it changes to elementary as a result of Process Synthesis.

Select the entity type or subtype from the Data Model, then select the process. You can select whether or not to populate the views for each type of entity action.



You can also customize the way Process Synthesis executes by selecting from the following options:

**Review default actions**

This option allows you to select actions to be performed on related entity types for create, update, and delete entity actions.

If you do not select this option, Process Synthesis accepts the default values of actions to be performed on related entity types. See Process Synthesis Rules in this chapter for a description of the rules used when generating the Action Diagram statements.

**Consistency check on entity type**

If you select this option, Consistency Check before Process Synthesis continues. A pop-up stating that a report has been prepared indicates that a consistency check has found inconsistencies.

**Note:** For more information about consistency checks, see the Toolset Help.

If your model is inconsistent, Process Synthesis stops executing. To continue, you have two options:

- Correct the error conditions and re-execute Process Synthesis
- Execute Process Synthesis without Consistency Check

If you do not select this option, Consistency Check does not execute and you can proceed with Process Synthesis.

**Automatic attribute view selections**

If you select this option, Process Synthesis includes all basic and designed attributes of the required entity types. This includes the setup of import, export, and entity action views. CA Gen does not list attributes for selection.

If you do not select this option, a list of all attributes of the subject entity type and related entity types displays. The list allows you to select the attributes you want to include in the logic. For a description of the rules used when generating the Action Diagram statements, see Process Synthesis Rules.

The advantage of using Process Synthesis from within the Data Model is that you can easily review the entity types and relationships you are working with before initiating Process Synthesis. In the other tools, you select entity types from a list and must have a working knowledge of the relationships between the subject and the related entity types.

## Process Synthesis from Activity Hierarchy and Activity Dependency

To perform Process Synthesis from the Activity Hierarchy and Activity Dependency Diagrams, you must complete the Process Hierarchy Diagram or Process Dependency Diagram as far as the parent process. To generate processes, you detail the non-elementary parent process. The generated processes become subordinates of the process that you detail.

The Activity Hierarchy Diagram and Activity Dependency Diagram are the only tools from which you can generate both action dialog and process boxes that appear in the Process Hierarchy and Process Dependency Diagrams. In the Process Hierarchy Diagram, a generated process appears on the activity hierarchy as a subordinate of a non-elementary parent process. In the Process Dependency Diagram, you must expand the parent process to see the generated process.

The advantage of using Process Synthesis from within the Activity Hierarchy Diagram and Activity Dependency Diagram is that you can generate five processes at one time, reflecting five types of action logic against the subject entity type.

The generated views and definition properties appear in the Process Hierarchy Diagram, Process Dependency Diagram, and Action Diagram. CA Gen indicates through expected effects in the Process Hierarchy Diagram and Process Dependency Diagram whether the process creates, reads, updates, or deletes an entity type.

## Process Synthesis from Action Diagram

To perform Process Synthesis with the Action Diagram tool, you must access a Process Action Diagram, Procedure Action Diagram, or an action block that has no action statements. You have several options for accessing Process Synthesis in the Action Diagram:

- Within the Activity Hierarchy Diagram, Activity Dependency Diagram, Structure Chart, or Action Block Usage, access an elementary process that contains no statements.
- From Dialog Design, Screen Design, Prototyping, Structure Chart, or Action Block Usage, access (chain to) a Procedure Step Action Diagram that contains no action statements.
- Within the Process Action Diagram:
  - Create a new Action Diagram or action block
  - Select an elementary process, Procedure Step, or action block

- Access another Action Diagram
- Access an empty action block from a USE statement in an Action Diagram or action block

After you select the Action Diagram or action block, you select the entity type or subtype followed by the type of action dialog you want to generate. You can then customize the way Process Synthesis executes. You can select from the same options that are available when you perform Process Synthesis with the Data Model:

- Review default actions
- Consistency check on entity type
- Automatic attribute view selections

**More information:**

[Perform Process Synthesis](#) (see page 132)

## Review Results of Process Synthesis

To see the results of Process Synthesis from the Activity Hierarchy Diagram, Activity Hierarchy Diagram, and Action Diagram, review the following generated results:

- Processes
- Definition properties for processes
- Expected effects
- Views
- Action Diagrams

## Review Generated Processes

After Process Synthesis from the Activity Hierarchy Diagram or Activity Dependency Diagram, the generated process boxes appear in the Process Hierarchy Diagram and Process Dependency Diagram.

## Review Generated Definition Properties

Process Synthesis assumes that the process to be generated is elementary and is non-repetitive. Both assumptions are explicit in the definition properties generated for each process. You can view these properties by detailing properties in the Activity Hierarchy Diagram.

## Review Generated Expected Effects

After Process Synthesis, you can review expected effects in the Process Hierarchy Diagram and Process Dependency Diagram to see whether the subject entity type and related entity types are created, read, updated, or deleted.

For example, if you generate logic for the process Create Order based on the Data Model shown in the Take Order Process Example, expected effects for the process would reflect the following actions:

- CREATE order
- CREATE order line
- READ product
- READ customer

When you perform Process Synthesis against a process with previously defined expected effects, the new expected effects are added. Existing effects are not changed. For example, if you had any expected effects of UPDATE defined for the entity types in the previous example, they would still appear.

## Review Generated Views

Process Synthesis lets you choose manual or automatic selection of attribute views. If you choose automatic selection, Process Synthesis automatically creates and populates views of all basic and designed attributes that are used in the action statements. If you choose manual selection, the same types of views are created. However, you select the attributes of the views used for READ statement qualifiers in the delete, read, and list processes, and the attributes of the views used for SET statements in the create and update processes.

If the entity types against which you are generating actions have any relationship memberships defined with a cardinality of one or more, a repeating group view is created.

In the Process Action Diagram, Process Synthesis populates the appropriate views for the action statements.

Process Synthesis does not create SET statements for derived attributes. This is because the SET statement for each derived attribute appears in its derivation algorithm, which is invoked by database accesses (I/Os) for the derived attribute.

**Note:** For more information about derived attributes, see the *Analysis Tools Reference Guide*.

Further, Process Synthesis does not create SET statements for auto number attributes. This is because the DBMS used by the application will automatically populate this column as a row is inserted into the table.

The generated views have default names which are shown in the following table. It includes names for additional view occurrences. You can rename views and add names to the entity action views by accessing view maintenance in the Activity Hierarchy Diagram, Activity Dependency Diagram, or Process Action Diagram. The following table displays Default Names for Generated Views:

View Type	Import View Subset	Export View Subset	Entity Action View Subset
Entity View	IMPORT	EXPORT	<Unnamed>
EV–2nd Occurrence	IMPORT_2	EXPORT_2	Persistent
Group view	GROUP_IMPORT	GROUP_EXPORT	N/A
GV–2nd Occurrence	GROUP_IMPORT_2	GROUP_EXPORT_2	N/A

If you perform Process Synthesis against a process with previously defined views, Process Synthesis deletes the current views and creates new ones as needed.

## Review Generated Action Diagrams

Each generated Process Action Diagram contains the following:

- Views
- One or more MOVE statements
- One or more entity action groups
- Branches for exception logic, such as *WHEN already exists* or *WHEN not unique*

The generated MOVE statements move the input view or the subject entity action view of the entity type to the output view. In the create and update Action Diagrams, statements that move the input view to the output view appear in the exception logic for the create or update action. Placing the MOVE statement after the action statements displays user input in the export view only after the process executes successfully. The list Action Diagram has one MOVE statement appearing at the end to move the entity action view to the export, which is a repeating group view.

Process Synthesis always populates the *WHEN successful* branch of exception logic with additional actions to perform. It does not populate the branches of exception logic (*WHEN already exists*, *WHEN not found*, and *WHEN not unique*) with additional actions.

## Review Generated PAD for CREATE

The generated create Action Diagram creates an occurrence of the entity type you specify during Process Synthesis. In addition, it may create or read related entity types as specified in data navigation.

The following example shows a generated Process Action Diagram for CREATE CUSTOMER:

```
CREATE_CUSTOMER
IMPORTS:...
EXPORTS:...
LOCAL:
ENTITY ACTIONS:...
CREATE customer
SET name TO input customer name
SET number TO input customer number
SET address TO input customer address
SET city TO input customer city
SET state TO input customer state
SET zip TO input customer zip
WHEN successful
MOVE customer TO output customer
WHEN already exists
```

Import, export, and entity action views are present, indicated by ellipses (...). Local views are not included; you must add them as needed for business rules and specific implementations. In this example, all views represent the subject entity type CUSTOMER. Views of any related entity types do not appear in the example of the Generated CREATE CUSTOMER Process Action Diagram because those entity types were not selected during Process Synthesis.

The MOVE statement after the WHEN successful exception logic executes only if the create succeeds. By moving import to export after execution of the process, the keyed input displays as output on the original screen only if the process succeeds.

The exception logic following WHEN already exists contains no action statements when the process is generated. You may want to add statements to handle this case. Action statements you add after WHEN already exists execute if an entity type with the same identifier has already been created.

Multiple associations and related entity types are shown in the example of a generated Process Action Diagram for CREATE CUSTOMER ORDER.

```
CREATE_CUSTOMER_ORDER
IMPORTS: Group View group_import
Entity View import customer_order
Entity View import customer
EXPORTS: Group View group_export
Entity View export customer_order
Entity View export customer
LOCAL:
ENTITY ACTIONS: Entity View customer_order_line
Entity View product
Entity View customer_order
Entity View customer
READ customer
WHERE DESIRED customer number IS EQUAL TO input
customer number
WHEN successful
MOVE customer TO output customer
CREATE customer_order
ASSOCIATE WITH customer WHICH places IT
SET number TO input customer_order number
SET date TO input customer_order date
SET status_code TO input customer_order status_code
SET confirmed_date TO input customer_order confirmed_date
SET time TO input customer_order time
WHEN successful
MOVE customer_order TO output customer_order
FOR EACH supplied
TARGETING
READ product
WHERE DESIRED product number IS EQUAL TO input
product number
WHEN successful
MOVE product TO output product
```

```
CREATE customer_order_line
ASSOCIATE WITH product WHICH ordered_on IT
ASSOCIATE WITH customer_order WHICH contains IT
SET number TO input customer_order_line number
SET quantity TO input customer_order_line quantity
SET requesting_unit TO input customer_order_line
requesting_unit
SET status_code TO input customer_order_line status_code
WHEN successful
MOVE customer_order_line TO output customer_order_line
WHEN already exists
WHEN not found
WHEN already exists
WHEN not found
```

The import, export, and entity action views show views of the subject entity type and related entity types.

The expression in the READ statement is the identifying attribute of the input view of the related entity type. If multiple identifying attributes are specified for the entity type, they also appear in the READ statement.

Generated Process Action Diagrams can handle any number of related entity types. For each related entity type, Process Synthesis generates the necessary views, READ (or CREATE), and ASSOCIATE statements. In fact, the CREATE action block appears under the WHEN successful branch of the READ for the related entity type. The second exception branch of the READ statement is WHEN not found.

The create action block uses SET statements to set the attributes of the entity action view to the corresponding input attributes, if the attributes are basic or designed.

You can add exception logic in this diagram or wait to add it during Design, after Process Synthesis. If the exception logic requires the execution of another process, you may want to wait until Design, when you can specify dialog flows and define exit states.

## Review Generated PAD for READ

The generated read Action Diagram uses specific selection criteria to retrieve each occurrence of the subject entity type and places its contents in an entity action view.

The generated Process Action Diagram for the read action is the same as that for the delete action except that it does not have the DELETE statement. If the subject entity type is related to other entity types, the read Action Diagram generates views for the related entity types and includes the related entity types in the READ statement.

Using Process Synthesis to generate the read Action Diagram is especially helpful when the subject entity type is related to multiple entity types and you want Process Synthesis to follow them for you.



The following is an example of a generated Process Action Diagram for READ CUSTOMER:

```
READ_CUSTOMER
IMPORTS: Entity View import customer
EXPORTS: Entity View export customer
LOCAL:
ENTITY ACTIONS: Entity View customer
READ customer
WHERE DESIRED customer number IS EQUAL TO input
customer number
WHEN successful
MOVE customer TO output customer
WHEN not found
```

Import, export, and entity action views are views of the subject entity type. In this example, if a related entity type was included, import, and entity actions views for that entity type would also appear.

The attribute view number is the identifying attribute of the subject entity type.

The MOVE statement following the WHEN successful branch moves all the attributes of the subject entity type to the output view.

The branch for the exception logic, WHEN not found, contains no action statements when the process is generated. You should add the processing necessary to handle this condition.

## Review Generated PAD for UPDATE

The generated update Action Diagram reads and then modifies each occurrence of the subject entity type. The diagram also reads each related entity type that you have specified.

TRANSFER statements appear for transferable relationships if you have selected this option during Process Synthesis.

If you have indicated disassociate for related entity types, READ EACH statements are generated for each entity type involved in relationships with the subject entity type where the destination has a cardinality of *many*. All occurrences will then be disassociated.

The following example shows the generated Process Action Diagram for UPDATE CUSTOMER.

The following example is an UPDATE CUSTOMER Generated Process Action Diagram:

```
UPDATE_CUSTOMER
IMPORTS: Entity View import customer
EXPORTS: Entity View export customer
LOCAL
ENTITY ACTIONS: Entity View customer
READ customer
WHERE DESIRED customer number IS EQUAL TO input
customer number
WHEN successful
UPDATE customer
SET name TO input customer name
SET address TO input customer address
SET city TO input customer city
SET state TO input customer state
SET zip TO input customer zip
WHEN successful
MOVE customer TO output customer
WHEN not unique
WHEN not found
```

Import, export, and entity action views are views of the subject entity type in this example.

The MOVE statement after the exception logic WHEN successful executes only if the read succeeds. By moving import to export after execution of the process, the keyed input displays as output on the original screen only if the process succeeds.

The READ statement selects the correct subject entity type by its identifying attribute or attributes.

The update action has two branches for exception logic: WHEN successful and WHEN not unique. The UPDATE action statements follow the WHEN successful branch of the READ statement.

Action statements following the WHEN not unique branch are not populated. You need to add them to specify what logic occurs if the subject entity type is found to have the same identifier as another entity type.

After the WHEN not found branch of the read, specify what to do if the specific occurrence of the subject entity type does not exist.

## Review Generated PAD for DELETE

The generated delete Action Diagram reads each occurrence of the subject entity type and removes it from the database. It also contains READ statements for each related entity type that you have specified.

The following example shows a generated Process Action Diagram for DELETE CUSTOMER. The following example is a DELETE CUSTOMER Generated Process Action Diagram:

```
DELETE_CUSTOMER
IMPORTS: Entity View import customer
EXPORTS: Entity View export customer
LOCAL:
ENTITY ACTIONS: Entity View customer
READ customer
WHERE DESIRED customer number IS EQUAL TO input
customer number
WHEN successful
MOVE customer TO output customer
DELETE customer
WHEN not found
```

In this example, import, export, and entity action views are views of the subject entity type. The attribute view is the identifying attribute of the subject entity type.

If the read of the subject entity type is successful, the Action Diagram moves all attributes of the subject entity type CUSTOMER to the output view and deletes them.

The Action Diagram contains two entity actions, a read and a delete. The read has two branches for exception logic: WHEN successful and WHEN not found. If the read of the subject entity type is successful, the entity action view indicated in the MOVE statement following the WHEN successful branch is moved to output and deleted. Before the export view is deleted from the database, you may want to archive the data, perhaps on tape. If so, add an external action block for this purpose during Design.

If the subject entity type references an entity of another type, the generated delete also builds the necessary views for the related entity type and adds a WHERE clause to the basic READ statement.

As always, a delete action deletes all entity types paired with the subject entity type by a mandatory relationship. For example, if you delete occurrences of INVOICE, all associated occurrences of INVOICE LINE are also deleted.

## Review Generated PAD for LIST

The generated Action Diagram containing the list (READ EACH) action logic reads each occurrence of the subject entity type.

Because of the assumptions made in the targeting clause, the list action dialog is the most likely of the generated processes to require modification. However, it can serve as a template and is useful in generating views.

The following example shows a generated Process Action Diagram for LIST CUSTOMER.

LIST CUSTOMER Generate Process Action Diagram

```
LIST_CUSTOMER  
EXPORTS: Group View group_export  
ENTITY ACTIONS: Entity View customer  
READ EACH customer  
TARGETING presented FROM THE BEGINNING UNTIL FULL  
MOVE customer TO output customer
```

The export and entity action views are views of the subject entity type.

The MOVE statement moves the input view to the output view. This move preserves user input in case the process fails to execute successfully.

The TARGETING clause specifies a repeating group export view that is populated each time the READ EACH performs an iteration. After each iteration that moves an entity type view subordinate to the repeating group view, the Action Diagram primes the repeating group view to accept the next entity type view subordinate.

The UNTIL FULL clause causes the READ EACH process to terminate if it attempts to add more occurrences to the repeating group view than were specified as the maximum cardinality of the group view.

**Note:** Process Synthesis gives the group view a default maximum cardinality equal to the maximum occurrences specified in the entity type properties (in the Data Model) or a value of 10 if maximum occurrence was never specified.

# Chapter 9: Structure Chart and Action Block Usage Tools

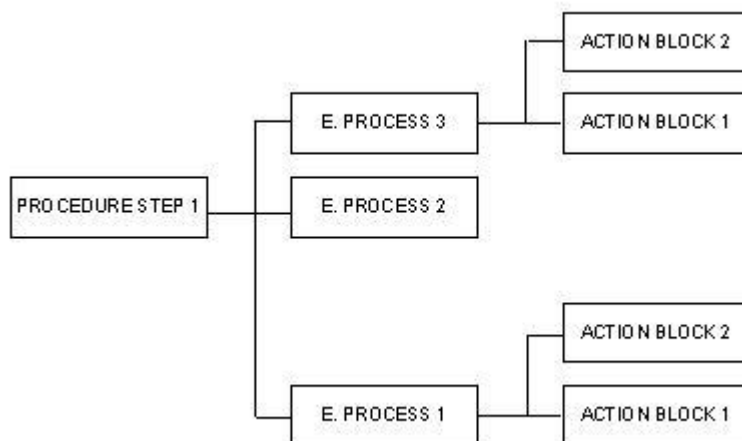
---

Two related tools, the Structure Chart and the Action Block Usage, graphically display the context of action blocks in your model. An action block is a standalone Action Diagram that other diagrams use. During Analysis you can use both tools for the interaction analysis task to refine process logic. During Design you can use both tools for the procedure logic design task to refine procedure logic.

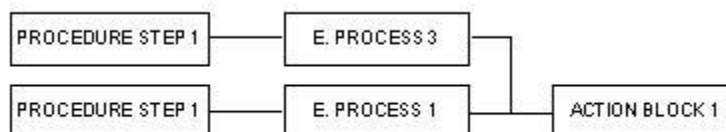
The Structure Chart displays a diagram of the hierarchy of action blocks used by a selected process, procedure, or action block. The Action Block Usage tool displays a diagram that shows where an action block is used in processing.

The following illustration shows the organization of a Structure Chart and an Action Block Usage Diagram, respectively. The term E. PROCESS refers to an elementary process.

This diagram shows the organization of a Structure Chart:



This diagram shows the Organization of an Action Block Usage Diagram:



## Structure Chart

You use the Structure Chart to analyze Action Diagrams. The branches of a Structure Chart represent the paths to lower level action blocks referenced by the USE command in the Action Diagram, including special functions.

CA Gen creates the diagram automatically when you access the Structure Chart and select an Action Diagram name. The Structure Chart consists of the selected process, procedure, or action block (represented by a rounded-corner rectangle) displayed in the middle of the screen, with the hierarchy of called action blocks (if any) branching from it.

The following colors distinguish objects on a structure chart:

Color	Diagram Object
White	Action block associated with a Procedure Step
Blue	Action block associated with an elementary process created during Analysis.
Magenta	Action block created with the Action Diagram tool
Cyan Outline	Special (IEF_supplied) functions referenced by a Procedure Step or action block

When you access the Structure Chart from Analysis, you can display the charts of a process or an action block. When you access the tool during Design, you can display the Structure Chart of a Procedure Step or an action block.

A Structure Chart provides the following benefits:

- Provides information quickly about the structure of an Action Diagram
- Eliminates the need to chain between Action Diagrams to review the structure
- Provides a simple way to match import and export views of action blocks called by the USE command
- Allows you to see Procedure Step action block decomposition in a different way

Action blocks are created during Analysis and Design:

- When adding common action blocks using the Action Diagram tool during Analysis and Design
- When automatically generating a process using the Activity Hierarchy Diagram or Data Model tool during Analysis
- When implementing a process in a procedure using the Dialog Design tool during Design

## How to Access the Structure Chart

Before using the Structure Chart, you must invoke CA Gen and access an existing model.

You can access the Structure Chart from the following sources:

- Analysis action bar
- Design action bar
- Activity Hierarchy Diagram
- Activity Dependency Diagram
- Dialog Design
- Screen Design
- Prototyping
- Action Diagram
- Action Block Usage

## Action Block Usage Tool

You use the Action Block Usage tool to analyze the usage of action blocks. An Action Block Usage diagram shows how an action block has been referenced through the USE command by other action blocks, processes, and Procedure Steps. This is similar to an index reference in a book. The branches of an Action Block Usage diagram represent the paths from each higher level action block to the same lower level action block.

CA Gen creates this diagram automatically when you access the Action Block Usage tool and select an Action Diagram name. The diagram displays the selected action block on the right, connected to the processes, Procedure Steps, or action blocks that call it, on the left.

The following colors are used to distinguish objects on an Action Block Usage diagram:

Color	Diagram Object
White	Procedure Step
Blue	Elementary process (created during Analysis)
Magenta	Action block added during Analysis or Design

Action Block Usage diagrams provide the following benefits:

- Provides information quickly about how an action block is used in the model
- Eliminates the need to chain between Action Diagrams to review the structure
- Provides a simple way to match import and export views of action blocks called by the USE command
- Allows you to see Action Block Usage in different ways

Action blocks are created during both Analysis and Design:

- When adding common action blocks using the Action Diagram during Analysis and Design
- When automatically generating a process using the Activity Hierarchy Diagram or Data Model during Analysis
- When implementing a process in a procedure using the Dialog Design during Design.

## Access the Action Block Usage Tool

Before using Action Block Usage, you must invoke CA Gen and access an existing model.

- Defined one or more business systems during Analysis
- Created Procedure Steps during Dialog Design

**Note:** For more information about defining a business system and defining Procedure Steps, see the *Design Guide*.

You can access Action Block Usage in the following ways:

- Analysis action bar
- Design action bar
- Activity Hierarchy Diagram (in Analysis only)
- Activity Dependency Diagram (in Analysis only)
- Structure Chart
- Action Diagram



## Using a Structure Chart or Action Block Usage Diagram

After you have displayed the Structure Chart or Action Block Usage diagram, you can do the following tasks:

- Add an action block
- Detail an action block

The actions you take upon action blocks are reflected in the Action Diagrams associated with the graphic objects.

### Add an Action Block

You can add action blocks to Structure Charts or Action Block Usage diagrams. The parent can be an action block, a process, or a Procedure Step.

After you add an action block, you cannot delete it using the Structure Chart or Action Block Usage tool.

The following criteria are used for an action block to be on the delete list:

- Not protected
- Not used by another action block (No REFDBY association)

If the action block is associated with an elementary process (DTLOGT association), you must use Analysis—Activity Hierarchy to delete the action block and its associated processes.

### Detail an Action Block

Detailing action blocks (except special functions) involves the following tasks:

- Changing the name
- Describing the action block
- Accessing view maintenance
- Matching views, including matching an import view with a supplying action block or matching an export view with a receiving action block

You cannot perform any activities on action blocks that represent special functions.

### Changing the Name of an Action Block

You may rename any action block shown on the Structure Chart or Action Block Usage diagram. The change is applied to both the chart or diagram and the Action Diagram.

## Describing an Action Block

You can add a description to the action block that explains its purpose and relevance to other action blocks.

## Accessing View Maintenance

You can access view maintenance for an action block.

### More information:

[Introduction to Views](#) (see page 89)

## Matching Views

The Structure Chart and Action Block Usage tools let you match views. CA Gen guides you through view matching by providing possible views to match and determining whether the match is possible.

You can match the import and export views of the selected action block.

## Using Other Action Options

The Structure Chart and Action Block Usage tools let you perform the following additional tasks:

- Chain to another tool
- Check the consistency of an action block
- Contract, expand, and redraw a Structure Chart or Action Block Usage diagram
- Unmatch views

## Chain

From the Structure Chart, you can link from an action block to the Action Diagram, to the Structure Chart (showing lower levels of the chart, if any), or the Action Block Usage diagram. You can also link from a Procedure Step to the Screen Design, Dialog Design, Prototyping tool, or the Action Diagram.

From Action Block Usage, you can link from an action block to the Action Diagram, to the Structure Chart (showing lower levels of the chart, if any), or the Action Block Usage Tools. You can also link from a Procedure Step to the Screen Design, Dialog Design, Prototyping tool, or the Action Diagram.

## Check

The Structure Chart and Action Block Usage tools allow you to check the consistency of a process, procedure, or action block.

## Contract

The Structure Chart and Action Block Usage tools allow you to contract the levels of action blocks that are visible. The higher the level of action block you select to contract, the more the chart is contracted.

## Expand

After you have contracted the levels of visible action blocks, you can expand the diagram to show the next immediate level of action blocks.

## Expand All

The Structure Chart and Action Block Usage tools let you expand all subordinate action blocks that may be contracted.

## Redraw

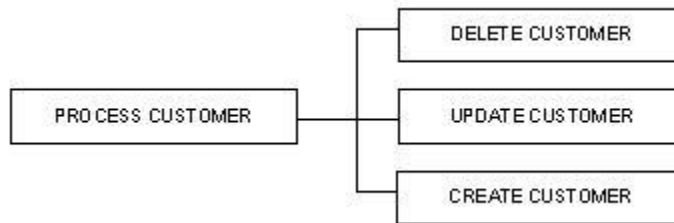
Both the tools, Structure Chart and Action Block Usage, let you redraw the Structure Chart or Action Block Usage diagram in different ways.

You have the following display options for Structure Chart and Action Block Usage Tools:

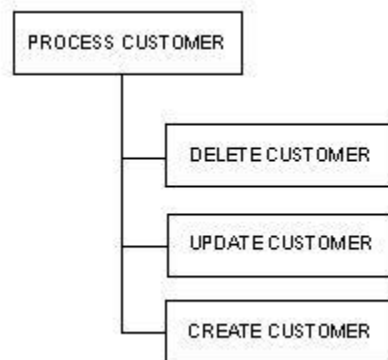
Option	Description
Box height	Lets you display an action block as a box with 1, 2, or 3 lines per box
Inverted display	Lets you toggle the direction of the chart.
Chart/diagram type	Lets you display an action block as a box with 1, 2, or 3 lines per box.

After you select a diagram placement scheme, CA Gen immediately redraws the chart according to your selections, as shown in the following figures.

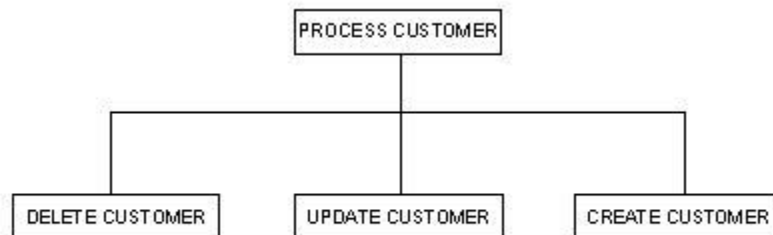
This diagram shows a Horizontal Structure Chart:



This diagram shows an Indented List Structure Chart:



This is a Vertical Structure Chart:



## Unmatch

The Structure Chart and Action Block Usage tools allow you to unmatch views that have been mistakenly matched.

# Chapter 10: Procedure Synthesis

---

This chapter discusses Procedure Synthesis, a task you conduct during procedure definition in Design.

In Design, you use the information discovered during Analysis as the basis for describing an information system (referred to as a *business system*) that satisfies the needs of the business. The objective of Design is to define the human-to-computer interactions required to perform the business activities defined during Analysis.

In procedure definition, you determine what kinds of procedures are required to implement the elementary processes discovered during Analysis. A procedure defines *how* business activities will be conducted. In essence, a procedure is the method by which one or more processes are carried out using a specific implementation technique. In a batch processing system, a procedure represents a single batch job.

In an online transaction processing system, a Procedure Step (a subdivision of a procedure) represents a transaction conducted at a single screen.

In a windowed or graphical user interface (GUI) system, a Procedure Step may represent the set of windows and dialog boxes that complete a task. A Procedure Step may also represent a separate client or server application in a client/server environment.

## Create a Procedure Action Diagram

After you decide which processes to implement, you must create a Procedure Action Diagram. You can use these methods to prepare this Action Diagram:

- Create the Procedure Action Diagram from scratch and manually construct views that combine the views of all selected processes. Building process logic from scratch requires you to select each command from a menu. This method may take more time to complete and assumes familiarity with the Action Diagram tool, but it offers great flexibility and control.
- Use Procedure Synthesis in the Design toolset to conduct synthesis (or resynthesis) which automatically builds a design based on analysis objects and design criteria selected. During Procedure Synthesis, CA Gen generates a Procedure Action Diagram that describes procedure logic. The Procedure Action Diagram USEs the Process Action Diagrams being implemented in the procedure. CA Gen also prepares a set of data views for the Procedure Action Diagram based on the combined views of all implemented processes.

The goal of Procedure Synthesis is to produce an Action Diagram that you can refine during procedure logic design to reflect your business needs completely and accurately.

**Downstream Effects:** The Construction toolset generates program code based on Procedure Action Diagrams. The Procedure Action Diagram uses the logic of the Process Action Diagrams for the processes implemented in the procedure.

The Construction toolset generates software that consists of screens populated with the fields initially defined with views. Code generation depends on the manipulation of views in Procedure Action Diagrams.

## Prerequisites

The prerequisites for Procedure Synthesis are listed next:

- Completion of elementary processes (dependent upon options in Process Synthesis)
- Definition of a business system

Completion of the Business System Definition is the last task in Analysis. This task reveals the list of elementary processes that are necessary for the business system.

## How to Access Procedure Synthesis

Access Procedure Synthesis in Dialog Design after adding and detailing a Procedure Step.

# Chapter 11: Using Procedure Synthesis

---

During business system design you must consider how the user will use the generated system. For online systems, you use Procedure Synthesis to determine how the user interacts with screens that present the generated software.

When you conduct Procedure Synthesis, you can perform the following tasks:

- Select the stereotype
- Select stereotype options
- Synthesize flows
- Resynthesize processes

The remainder of this chapter discusses these topics.

When you select the stereotype you can also select options that influence both procedure definition and dialog design. For example, you can assign commands when you implement action blocks. Command is a special attribute. It allows you to specify the means for users to influence procedure execution. In an online system, you can place a command field on a screen. At execution time, the user can enter a value into the command field to direct execution of the procedure. For example, consider a procedure called Maintain Customer that implements elementary processes called Add Customer, Change Customer, and Delete Customer. If you specify the command to be used for execution, the user can select among the three processes by placing the appropriate value in the command field. During Procedure Synthesis you can set the command in a Procedure Action Diagram that executes the required action block.

## Select the Stereotype

Building Procedure Action Diagrams consists primarily of selecting and applying a stereotype. A stereotype refers to a typical style of procedure design. In essence, a stereotype is a pre-defined implementation strategy for one or more business processes. When you apply a stereotype, CA Gen generates a stereotypical Procedure Action Diagram. You select the options that influence the design of the Action Diagram. After it is created, you can refine the procedure logic to reflect the needs of the business system. If necessary, you can repeat stereotyping. During each stereotyping iteration, you can preserve changes you made as a result of the last iteration or replace the entire Procedure Action Diagram with code from the current iteration.

You can select one or more of the following stereotype styles:

- Entity maintenance stereotype
- Selection list stereotype
- Stereotype by implementation of action blocks

These stereotypes are designed for block mode (online transaction processing) screen systems. Most parts are adaptable to GUI systems.

**Downstream Effects:** The major benefit of stereotyping besides automation of implementation is reusability, reusing typical procedure definitions for any procedure.

Reusability has the following benefits:

- Improves quality
- Reduces overall project costs
- Encourages the use of corporate standards
- Encourages the commonality of user interfaces
- Reduces corporate risk

## Entity Maintenance Stereotype

An entity maintenance stereotype synthesizes a procedure for entity maintenance actions. You select an action block for each of the four entity actions (create, read, update, and delete). You must select an action block to display (read) records. In an online system, you should read a record before you can update or delete it.

You also select a screen identifier from the import attribute views of the process that you selected to read records. CA Gen uses the screen identifier as the criteria for record searches.

Procedure Action Diagrams that result from entity maintenance stereotyping contain NOTE statements that signal you where to add custom validation logic.

A need for common logic often occurs at the end of a Procedure Action Diagram. To allow for this, error conditions found before the main CASE OF COMMAND statement do not escape out of the procedure. Instead, the condition sets the command to the value of BYPASS and stores away the actual command. A case of BYPASS occurs in the main CASE OF COMMAND statement to set the command back to the original value.

In addition, any ESCAPE statement in the main CASE OF COMMAND construct escapes only to the outside of the case structure. You can put any common logic at the bottom of the Procedure Action Diagram after the CASE OF COMMAND construct.



All views that Procedure Synthesis generates are set to optional. In addition, Procedure Synthesis generates a local work set attribute view named IEF\_SUPPLIED that stores the COMMAND.

## Selection List Stereotype

A selection list stereotype allows you to generate a Procedure Action Diagram that presents a selection list. A selection list shows the occurrences of a subject entity type. Each display line in the list displays one or more of the attributes. The occurrences are sorted by a selected attribute. During Procedure Synthesis, CA Gen creates a Procedure Action Diagram that describes the logic to display all occurrences of the subject entity type. CA Gen also creates a field to permit the user to enter a selection.

The following is an example of a selection list.

```
LIST OF EMPLOYEES
EMPLOYEE NUMBER: 0000000
NAME NUMBER
M. ADAMS 0198395
J. BROWN 1388404
W. COLLINS 9873913
R. JOHNSTON 5467938
T. NASH 9085940
E. ROBERTSON 9851098
C. SANDERS 2992847
D. SMITH 0999800
B. WALTERS 5986070
```

CA Gen automatically adds the CASE command to display a selection list.

Selection list stereotypes use the generated Dialog Manager for scrolling.

## Implementation of Action Blocks Stereotype

A stereotype based on the implementation of action blocks allows you to generate a Procedure Action Diagram that calls any action block. You can select action blocks for implementation from Analysis and Design.

You can use this type of stereotype for procedures that do not fit the entity maintenance or selection list stereotypes but can still benefit from automatic view synthesis. When you select the action block, you also select the command value that directs processing.

## Select Stereotype Options

In preparation for transformation, you can select options that modify the stereotype. Each modification results in annotated action statements in the Procedure Action Diagram.

You can do the following tasks:

- Add an action bar for command entry
- Execute entity actions in one- or two-stages
- Use the Enter command to update
- Use the Enter command to display
- Refresh the screen when the command is CANCEL
- Clear data from the screen

You can select all of these options for the entity maintenance stereotype after you select the action block that displays records and the screen identifier. You can select only the first option for the selection list stereotype after you select the subject entity type.

The following discussions explain these options.

### Add an Action Bar for Command Entry

A user can execute commands in three ways:

- Press a function key to which a specific command is assigned. See Business System Defaults in *Design*.
- Enter a command or synonym value in the command field
- Enter a one- or two-character code in the generated action entry field of an action bar and press a key that executes the command

You can have Procedure Synthesis add an action bar to a screen when you select a selection list stereotype or an entity maintenance stereotype. When you apply the stereotype, Procedure Synthesis generates an Action Diagram that includes the logic to list commands on the action bar and provide a field for command entry. The letters D, C, and U direct the system to Display, Create, and Update, respectively. Since Display and Delete both start with the letter D, you can distinguish the Delete command with the two-character entry DE.

The following illustration is an example of a screen with a generated action bar.

<u>DE</u>	C - Create	U - Update	D - Display	DE - Delete
Department Maintenance				
Department:				
Number				1
Name				Personnel

Procedure Synthesis generates the following statements when you choose to add an action bar for command entry:

```

NOTE Translate the Action Bar into a command
IF COMMAND IS EQUAL TO enter
  AND import ief_supplied action_entry IS NOT EQUAL TO SPACES
  SET temp ief_supplied command TO display
  IF substr(import ief_supplied action_entry, 1,1) IS EQUAL TO
    substr(temp ief_supplied command, 1,1)
    AND (substr(import ief_supplied action_entry, 2,1) IS EQUAL TO
      substr(temp ief_supplied command, 2,1)
    OR substr(import ief_supplied action_entry, 2,1) IS EQUAL TO
      SPACES)
  COMMAND IS display
  SET temp ief_supplied command TO update
  IF substr(import ief_supplied action_entry, 1,1) IS EQUAL TO
    substr(temp ief_supplied command, 1,1)
    AND (substr(import ief_supplied action_entry, 2,1) IS EQUAL TO
      substr(temp ief_supplied command, 2,1)
    OR substr(import ief_supplied action_entry, 2,1) IS EQUAL TO
      SPACES)
  COMMAND IS update
  SET temp ief_supplied command TO create
  IF substr(import ief_supplied action_entry, 1,1) IS EQUAL TO
    substr(temp ief_supplied command, 1,1)
    AND (substr(import ief_supplied action_entry, 2,1) IS EQUAL TO
      substr(temp ief_supplied command, 2,1)
    OR substr(import ief_supplied action_entry, 2,1) IS EQUAL TO
      SPACES)
  COMMAND IS create
  SET temp ief_supplied command TO delete
  IF substr(import ief_supplied action_entry, 1,1) IS EQUAL TO
    substr(temp ief_supplied command, 1,1)
    AND (substr(import ief_supplied action_entry, 2,1) IS EQUAL TO
      substr(temp ief_supplied command, 2,1)
  COMMAND IS delete
  IF COMMAND IS EQUAL TO enter
    MAKE export ief_supplied action_entry ERROR
    SET temp ief_supplied command TO COMMAND
    COMMAND IS bypass
  EXIT STATE IS action_bar_is_invalid

```

If you generate an action bar, you do not need to assign the same commands to PF (function) keys. In addition, you should not use the Enter key to display or update options. See [Use the Enter Command to Display](#) and [Use the Enter Command to Update](#).

You can also obtain the action bar functionality by using Command Synonyms and placing a command field on the screen. You may need to shorten the command field.

## Execute Entity Actions in One or Two Stages

In the simplest implementation of entity maintenance, the action executes immediately after the user issues the command. This is known as a one-stage execution. However, to reduce the risk of unintentionally executing entity actions that affect the database, you can have Procedure Synthesis generate Action Diagram logic that requires a two-stage execution:

- In the first stage, the user enters a command and the system returns a request for confirmation of the action.
- In the second stage, the user confirms the action and the system executes it.

During confirmation, the screen is protected from input.

You can select one- or two-stage execution individually for the create, update, and delete entity actions when you select an entity maintenance stereotype. Two-stage execution is most commonly used for the delete entity action.

The following example shows the Action Diagram statements that Procedure Synthesis generates for a two-stage deletion:

```

    ⋮
    NOTE two-stage processing requires resetting the
      LAST COMMAND field so that it lasts one pass.
    IF COMMAND IS NOT EQUAL TO confirm
    SET export_last ief_supplied command to SPACES
    ⋮
    NOTE Validation CASE OF COMMAND
    CASE OF COMMAND
    CASE delete
    NOTE Add specific DELETE validation here. If an
      error is found, an EXIT STATE should be set.
    IF EXIT STATE IS NOT EQUAL TO processing ok
    SET temp ief_supplied command TO COMMAND
    COMMAND IS bypass
    ELSE
    EXIT STATE IS valid_delete
    SET export_last_2 ief_supplied command TO COMMAND
    COMMAND IS protect
    CASE confirm
    NOTE Set command to LAST COMMAND.
    IF import_last ief_supplied command IS EQUAL TO update
      OR import_last ief_supplied command IS EQUAL TO create
      OR import_last ief_supplied command IS EQUAL TO delete
    COMMAND IS export_last ief_supplied command
    SET export_last ief_supplied command TO SPACES
    NOTE Main CASE OF COMMAND
    CASE OF COMMAND
    ⋮
    CASE protect
    NOTE All fields are protected after successful validation.
    COMMAND IS export_last_2 ief_supplied command
    MAKE export customer name Protected Normal Intensity Normal Color
    MAKE export customer number Protected Normal Intensity Normal Color
    MAKE export customer address Protected Normal Intensity Normal Color
    MAKE export customer city Protected Normal Color Normal Intensity
    MAKE export customer state Protected Normal Intensity Normal Color
    MAKE export zip customer Protected Normal Intensity Normal Color
    MAKE export_hidden_id customer number Protected Normal Intensity Normal Color
    MAKE export_last ief_supplied command Protected Normal Intensity Normal Color
    MAKE export_last_2 ief_supplied command Protected Normal Intensity Normal Color
    ⋮
  
```

## Use the Enter Command to Update

You can have Procedure Synthesis generate Action Diagram logic that executes commands based on the contents of screen fields. You can determine that when the user presses Enter after changing the data in any field except the screen identifier fields, the screen *updates* the data. This assumes that the Enter key is assigned a command of ENTER.

The following example shows the logic if you choose to have the user press Enter to update the data:

```
NOTE The ENTER command is used to request an UPDATE.  
Convert ENTER to UPDATE if the key has not changed.  
IF COMMAND IS EQUAL TO enter  
AND import customer number IS NOT EQUAL TO  
import_hidden_id customer number  
COMMAND IS display
```

## Use the Enter Command to Display

You can determine that when the user presses Enter after changing the data in the screen identifier fields, a new screen appears that *displays* the selection identified by the screen identifier field. This assumes that the Enter key is assigned a command of ENTER.

The following example shows the logic if you choose to have the user press Enter to display the next selection:

```
NOTE The ENTER command is used to request a DISPLAY  
of the next selection. Convert ENTER to  
DISPLAY if the key has changed.  
IF COMMAND IS EQUAL TO enter  
AND import customer number IS NOT EQUAL TO  
import_hidden_id customer number  
COMMAND IS display
```

## Refresh the Screen on Cancel

You can determine that when the user requests the CANCEL command, the generated system refreshes the screen from the database. The data that refreshes the screen is based on the screen identifier from the previous display.

The following example shows the logic for the CANCEL command:

```

NOTE If command is CANCEL and the hidden
identifier is blank, escape before moving
Imports to Exports so that the screen is
blanked out.
IF COMMAND IS EQUAL TO cancel
AND (import customer number IS EQUAL TO 0
EXIT STATE IS processing_ok
ESCAPE
.
.
.
NOTE: Main CASE OF COMMAND
CASE OF COMMAND
.
.
.
CASE cancel
NOTE Passes the hidden identifier rather than the import identifier when the display
module is called.
USE dis_customer
WHICH IMPORTS: Entity View import_hidden_id customer
WHICH EXPORTS: Entity View export customer
.
.
.

```

Procedure Synthesis also creates a CASE CANCEL that USEs the action block that displays the data.

## Clear Data from the Screen

You can determine that the user can select a command that clears data from all screen fields.


The following example shows the logic if you choose to have the user enter a command to clear the screen:

Command to Clear the Screen Example

```

NOTE if command is CLEAR, escape before moving
Imports to Exports so that the screen is
blanked out
IF COMMAND IS EQUAL TO clear
EXIT STATE IS processing_ok
ESCAPE

```



## Synthesize Flows

A flow refers to the movement of control and, optionally, data between Procedure Steps. Procedure Synthesis lets you create special flows between the procedure to be generated and other procedures.

Synthesize flows consists of the following tasks:

- Synthesize menu flows
- Synthesize subject list flows
- Synthesize neighbor selection list flows

The following discussions explain each task.

### Synthesize Menu Flows

In an online system, a menu gives the user a way to navigate between procedures in the current business system or in another business system. Procedure Synthesis lets you create flows between Procedure Steps so that one Procedure Step serves as a menu to another.

Procedure Synthesis creates a link flow and a transfer flow. The CASE statements that execute the link are built when the link flow is created. Each link occurs through an EXIT STATE IS *link\_to\_procedure\_name* statement, where *procedure\_name* refers to the name of the destination Procedure Step.

The transfer to a menu Procedure Step is created to manage the following situations:

- When a Procedure Step is displayed from a clear screen
- When there is a flow from another Procedure Step

You can also map views to ensure that the correct data flows. The figure that follows is an example of a basic menu.

Menu Example

Menu Example

```
MENU
1 = EMPLOYEES:
2 = DEPARTMENTS
```

Enter selection:



Menu flows can be added to any Procedure Steps that must be connected to other Procedure Steps.

## Synthesize Subject List Flows

Subject list flow synthesis allows you to add a flow between a synthesized selection list Procedure Step and the current Procedure Step. The selection list Procedure Step has the same subject entity as the entity maintenance procedure. You also select the command that executes the flow.

The selection list to which you connect the flow is intended to be used as a menu of the subject entity type occurrences for the Procedure Step being detailed. For example, you can synthesize a flow between a selection list Procedure Step that lists employees and an entity maintenance Procedure Step that performs entity actions on employee personnel records. The intent is to select the subject entity from the selection list and perform entity maintenance actions on the subject entity. At the completion of entity maintenance, the user returns to the selection list.

Procedure Synthesis automatically creates a transfer and a link. A link flows from the selection list Procedure Step to the entity maintenance Procedure Step. A command of `DISPLAY` invokes the `Execute First` option. The entity maintenance Procedure Step returns from the link on a command of `RETURN` and passes no command with the `Execute First` option. The current subject view in the entity maintenance Procedure Step is matched back on the return so that the selection list Procedure Step starts from the current selection.

A transfer flows from the entity maintenance Procedure Step to the selection list Procedure Step. A command of `DISPLAY` invokes the `Execute First` option at the selection list. Since no data is matched, the selection list is displayed from the beginning.

## Synthesize Neighbor Selection List Flows

Neighbor selection list flow synthesis also allows you to add a flow from a current Procedure Step to a selection list Procedure Step. The selection list Procedure Step has any subject entity type, including that of the current Procedure Step. The flow is implemented as a link. You select the command that executes the flow.

The intent of the neighbor selection list flow is to allow the user to complete an action at one screen (including an entity maintenance screen) by selecting an occurrence from a selection list. The occurrence of the selected neighboring entity type should have an attribute that appears in the view set. For example, the user may need to complete a name at an entity maintenance screen. By adding a flow to a neighbor selection list, the user can select the name from the list to complete the field on the entity maintenance screen.

You select the neighboring entity type from the import views of the current Procedure Step. A command of DISPLAY passes down the flow with the Execute First option. No data is matched on the flow. The flow returns to the current Procedure Step on a command of RETURN. The flow passes no command with the Display First option. On return from the neighbor procedure, the data maps to the selected entity view.

On many systems, this is called the Prompt option and is tied to PFkey 4 per IBM Common User Access (CUA) standards.

## Resynthesize Processes

The resynthesis of elementary processes and stereotype options into a Procedure Step is automatic. You simply apply a stereotype you have selected.

On completion of resynthesis, you can refine the Action Diagram for the Procedure Step using the Action Diagram tool. You can review the interconnections between action blocks using the Structure Chart tool. Finally, you can generate the screen in the Screen Design tool by selecting the Auto Layout action.

# Chapter 12: Asynchronous Behavior

---

CA Gen applications using synchronous processing use the Procedure Step USE statement to request a cooperative flow. The Procedure Step USE statement operates as one atomic operation by sending a view, waiting for a response, and receiving a view.

Asynchronous processing decouples the request from the response, so it is necessary to use one statement to initiate the request, and then a different statement to retrieve the response. That is, you specify a view that the initiating request sends (an import view), and a view in the retrieving command (an export view) to receive the response.

**Note:** Some environments do not support asynchronous processing. See the *CA Gen Technical Requirements* document for information about supported environments.

The action language statements associated with asynchronous processing are:

- USE ASYNC—to initiate an asynchronous request.
- GET ASYNC RESPONSE—to retrieve an asynchronous response.
- CHECK ASYNC RESPONSE—to check whether a response is available.
- IGNORE ASYNC RESPONSE—to ignore a response.

**Note:** For information about how to include the asynchronous action statements in your model, see the *Toolset Help*. For more information about covering asynchronous processing within a CA Gen distributed processing application, see the *Distributed Processing – Overview Guide*.

## ASYNC\_REQUEST Work Set

In addition to the asynchronous action statements, each CA Gen model contains a Work Set to support asynchronous behavior. This work set type, named ASYNC\_REQUEST, allows each outstanding request to contain a unique request identifier, and is used to communicate aspects of a given asynchronous request between the requesting application component and the supporting runtime components. Each unique outstanding request within an application must be represented by a unique instance of a view of the ASYNC\_REQUEST work set.

Each instance of an ASYNC\_REQUEST work set view contains the following attributes:

Attribute	Size	Type	Description
ID	(8)	numeric data	The ID attribute is a unique identifier associated with a given request. The ID field is set by the runtime when the asynchronous request is accepted for processing.
REASON_CODE	(8)	text data	Reserved for future use.
LABEL	(128)	mixed text data	The LABEL attribute is a character string that is set by the application program. It provides the program with a way to identify the given request.
ERROR_MESSAGE	(2048)	mixed text data	The ERROR_MESSAGE attribute only has value if the associated request completed with an error. The ERROR_MESSAGE value contains a text description of the failure. This attribute contains the same text data as is displayed in the event of a server error or communications error.

## Async Action Statements

Each ASYNC statement is composed of a set of arguments and a set of WHEN clauses. The Action Diagram statement editor supplies each ASYNC statement with its full complement of WHEN clauses, each of which you can choose to retain or delete, depending on the process flow that you require at that point in your application.

See the description of each of the preceding ASYNC statements for the behavior that results if a particular WHEN clause is present or is deleted.

**Note:** For information about including the asynchronous actions in your model, see the *Toolset Help*.

## USE ASYNC

This command description includes:

- The syntax of the command
- A description of the parameters
- Related examples.

## Syntax

```
+--USE ASYNC target_pstep (Procedure Step)
| WHICH IMPORTS:...
| IDENTIFIED BY: viewname async_request
| RESPONSE HANDLING: POLL | NOTIFY EVENT | NO RESPONSE
| RESPONSE EVENT: eventhandlername
| RESPONSE SCOPE: PSTEP | GLOBAL
+--WHEN request accepted
+--WHEN request not accepted
```

The USE ASYNC statement is used to initiate an asynchronous cooperative flow. The clauses of the USE ASYNC statement communicate, to the supporting runtime code, how to process the corresponding response of the request.

As with a USE PStep, the USE ASYNC statement must specify an import view that identifies the data to be sent to the server PStep.

**Note:** A USE ASYNC statement can only be used to flow to a Procedure Step that is packaged in a server manager. The USE ASYNC command does not support a flow from a windowed client to a non-windowed client.

## Request Identifier (IDENTIFIED BY:)

Each asynchronous request must be uniquely identified by the requesting application. The IDENTIFIED BY clause conveys the specific ASYNC\_REQUEST view instance that will be used, by the runtime, to reference a specific asynchronous cooperative flow. The ID attribute of the specified ASYNC\_REQUEST view is populated by the supporting runtime when that runtime accepts the request.

## Conditional Clauses (WHEN)

The optional set of WHEN clauses on the USE ASYNC statement give the application developer control that is not available with a synchronous cooperative flow. Each of the WHEN clauses is included in the code when a USE ASYNC statement is constructed by the PAD editor. You can delete either or both of the WHEN clauses.

**WHEN request accepted**—The code contained in this WHEN clause is given control if the supporting runtime reaches a point in its processing where the request is considered accepted. Typically, this is the point where the runtime has sent the request to the target server using the send mechanism of the supporting Middleware. For example, if MQSeries is the transport, all the processing leading up to and including the MPUT of the request must be successful for this WHEN clause to be given control.

If this clause is not present, processing continues with the first statement following the USE ASYNC.

**WHEN request not accepted**—The code contained in this WHEN clause is given control if the supporting runtime encounters an error attempting to initiate the asynchronous cooperative flow. The ERROR\_MESSAGE field, of the specified ASYNC\_REQUEST view, contains a message explaining why the given request was not accepted for processing.

If deleted, the runtime terminates the initiating PStep. This process is similar to an error occurring during a failed USE Procedure Step statement. The currently executing code block is exited.

**Note:** A request must be successfully accepted before it can be referenced by other statements that are subsequently used to check or retrieve the response (GET ASYNC RESPONSE, CHECK ASYNC RESPONSE and IGNORE ASYNC RESPONSE).

## Response Handling

The RESPONSE\_HANDLING clause is used to set Asynchronous requests as POLL requests or NOTIFY EVENT requests.

### Poll

A RESPONSE HANDLING clause specified as POLL communicates, to the supporting runtime code, that the application will ask for response information rather than expect to receive it automatically. The application will complete an outstanding request using a GET ASYNC RESPONSE statement or by using an IGNORE ASYNC RESPONSE statement. The GET ASYNC RESPONSE statement supports both active and passive polling, as described in the Processing Options for Async Requests section.

### Notify Event

A RESPONSE HANDLING clause specified as NOTIFY EVENT indicates that the application is notified when a response to an event is received by the supporting runtime. The event notification is handled in a similar way to existing window events. You must first specify that the application wishes to be notified, by specifying a value of NOTIFY EVENT for the RESPONSE\_HANDLING clause on the USE ASYNC statement. Additionally, you must associate an event handler with each asynchronous request that specifies an event be raised. The event handler name is supplied via the RESPONSE\_EVENT clause on the requesting USE ASYNC statement. A unique event type is created for each given instance of the USE ASYNC statement. This unique event type is associated with the specified event handler that has been specified using the RESPONSE\_EVENT clause.

When you add a NOTIFY EVENT type USE ASYNC statement, you can create a new EVENT ACTION block, or select an existing EVENT ACTION already associated with the PStep.

**Note:** The runtime only provides notice that the request has completed. This does not imply that the request completed successfully.

As with other window events, the PStep can only receive a given event when it has at least one open window. The open window is needed to send the event to the event action block that has been defined to handle the specified event type. If the window is closed, the event notification is ignored, and the response is retained or discarded depending on the response scope of the call. That is, the response is discarded if the response scope is PSTEP, and retained if the scope is GLOBAL. For more information, see Response Scope.

It is the responsibility of the application designer to determine if the request fulfilled the desired actions, and to obtain any requested data. The application may also, periodically, poll for completion status. However, any given outstanding request can only be *completed* once.

If an application attempts to complete a given outstanding request from more than one area of processing logic (for example, from the Action Diagram or an event handler), each statement should expect that the designated request has already been completed.

For example, the logic doing the polling may complete the request prior to the event handler being scheduled. The logic in the event handler, assuming it attempts to complete the request, must be coded to cope with the fact that the request that causes the event handler to be driven might, in fact, already be complete.

### **No Response**

Response handling set to NO RESPONSE indicates that the application is not interested in retrieving the corresponding response of the request. Additionally, no error conditions are reported to the client. Once accepted, no indication of how the request was processed is returned to the requesting client application.

## **Response Scope**

By default, the scope of an asynchronous request is limited to the initiating PStep. All outstanding requests initiated by a PStep are implicitly ignored when the PStep exits. When operating in a GUI application, there may be cases, you may wish to retain the response so that it can be obtained by a different PStep executing within the same process. The scope of the response is indicated via the RESPONSE SCOPE clause specified on the USE ASYNC statement. RESPONSE SCOPE may be either PSTEP or GLOBAL, where PSTEP limits the scope to the initiating PStep, and GLOBAL indicates that the response is considered global to all Procedure Steps executing in the process.

If you write one application PStep to initiate an asynchronous cooperative flow, and another PStep to complete the flow, you must preserve the content of the ASYNC\_REQUEST view instance that you used to initiate the outstanding request. This is necessary so that the processing that completes the flow can specify the identifier in either the GET ASYNC RESPONSE statement or the IGNORE ASYNC RESPONSE statement.

Contrary to the cleanup provided for the default scope of PSTEP (that is, where the request is considered complete), the runtime does not implicitly complete an outstanding request where the response scope is marked as GLOBAL. You must include application logic that is responsible for completing requests that indicate their associated response have a GLOBAL scope. That is, the application logic must issue either a GET ASYNC RESPONSE or an IGNORE ASYNC RESPONSE to complete the request.

**Note:** The CA Gen Server Runtime only supports a RESPONSE SCOPE of PSTEP. That is, a response to an outstanding request cannot linger across invocations of a target server PStep. All asynchronous cooperative flows initiated by a server PStep (that is, server-to-server flows) must be completed explicitly by the server PStep code, or implicitly by the Server Runtime, prior to returning control to the invoking client application component.

## Examples

The Action Diagram of the Toolset helps construct the USE ASYNC statement. The following examples show the various types of USE ASYNC:

- An asynchronous request indicating that the application does not wish to receive the response associated with the request:

```
+--USE ASYNC target_pstep (Procedure Step)
| WHICH IMPORTS: ...
| IDENTIFIED BY: viewname async_request
| RESPONSE HANDLING: NO RESPONSE
+--WHEN request accepted
+--WHEN request not accepted
+--
```

- An asynchronous request indicating that the application wishes to Poll for the response, where the response is only available to the PStep initiating the request:

```
+--USE ASYNC target_pstep (Procedure Step)
| WHICH IMPORTS: ...
| IDENTIFIED BY: viewname async_request
| RESPONSE HANDLING: POLL
| RESPONSE SCOPE: PSTEP
+--WHEN request accepted
+--WHEN request not accepted
+--
```



- An asynchronous request indicating that the application wishes to Poll for the response, where the response is available to other processing executing within the process:

```
+--USE ASYNC target_pstep (Procedure Step)
| WHICH IMPORTS: ...
| IDENTIFIED BY: viewname async_request
| RESPONSE HANDLING: POLL
| RESPONSE SCOPE: GLOBAL
+--WHEN request accepted
+--WHEN request not accepted
+--
```

- An asynchronous request indicating that the GUI application wishes to be notified when the response becomes available to be obtained from the runtime. Additionally, this request indicates that the response is available only to the PStep that initiated the request.

```
+---USE ASYNC target_pstep (Procedure Step)
|   WHICH IMPORTS: ...
|   IDENTIFIED BY: viewname async_request
|   RESPONSE HANDLING: NOTIFY EVENT
|   RESPONSE EVENT: eventname
|   RESPONSE SCOPE: PSTEP
+ WHEN request accepted
+ WHEN request not accepted
+---
```

- This example also shows the EVENT ACTION block that is created when the USE ASYNC statement was created. The name of the EVENT ACTION is automatically assigned to be the same as the target\_pstep referenced on the USE ASYNC statement. If the application developer wishes to create a new EVENT ACTION (as opposed to using a previously defined EVENT ACTION) and if for some reason an EVENT ACTION already exists with the target-pstep name, the Toolset will create a unique name for the new EVENT ACTION (for example, adding a 1, 2, 3, and so on).

```
+--USE ASYNC target_pstep (Procedure Step)
| WHICH IMPORTS: ...
| IDENTIFIED BY: viewname ASYNC_REQUEST
| RESPONSE HANDLING: NOTIFY EVENT
| RESPONSE EVENT: target_pstep
| RESPONSE SCOPE: PSTEP
+--WHEN request accepted
+--WHEN request not accepted
+--
.
.
.
+--EVENT ACTION target_pstep
|
+--
```

- An asynchronous request indicating that the application wishes to be notified when the response becomes available to be obtained from the runtime. Additionally, this request indicates that the response is available to other processing that is executing within the process:

```
+--USE ASYNC target_pstep (Procedure Step)
| WHICH IMPORTS: ...
| IDENTIFIED BY: viewname ASYNC_REQUEST
| RESPONSE HANDLING: NOTIFY EVENT
| RESPONSE EVENT: target_pstep
| RESPONSE SCOPE: GLOBAL
+--WHEN request accepted
+--WHEN request not accepted
+--
.
.
.
+--EVENT ACTION target_pstep
|
+--
```

## GET ASYNC RESPONSE

This command description includes the following information:

- The syntax of the command
- A description of the parameters

### Syntax

```
+--GET ASYNC RESPONSE target_pstep (Procedure Step)
| WHICH EXPORTS:...(syntax the same as USE PStep)
| IDENTIFIED BY: viewname async_request
+--WHEN successful
+--WHEN pending
+--WHEN invalid ASYNC_REQUEST_ID
+--WHEN server error
+--WHEN communications error
```

When an application uses asynchronous processing, it is the responsibility of the application to obtain the response of the target PStep. The GET ASYNC RESPONSE statement allows the application to request a response. As with a synchronous USE PStep statement, the data must be mapped into export views so that the application can make use of the response data.

The behavior of the GET ASYNC RESPONSE may be modified, depending on whether the WHEN Pending clause is present on the statement.

- If the WHEN pending clause is coded, the application is requesting active polling. Control is returned to the application if the response is not immediately available.
- If the WHEN pending clause is not coded, the application is requesting passive polling. The GET ASYNC RESPONSE statement blocks until the response becomes available.

**Note:** Depending on the logic of the application, it may be more efficient to use the CHECK ASYNC RESPONSE statement, followed by the GET ASYNC RESPONSE statement. The check statement is simply a status check, and does not incur any of the resource overheads associated with the get response statement (for example, storage for the response message object). However, this processing may be offset by the increased processing time used if the application loops, repeating a check statement.

### Request Identifier (IDENTIFIED BY:)

The application must keep track of each outstanding request, so that it can obtain a corresponding response of a given request. The application indicates the response by the specified ASYNC\_REQUEST view. The specified ASYNC\_REQUEST Work Set view should be the same view (or at least a view that contains the same values) as was specified on the USE ASYNC statement that initiated the original asynchronous flow.

Due to the inherent, disjointed nature of processing asynchronous cooperative flows, the application logic may be engaged in other processing when a response to an outstanding request becomes available. The GET ASYNC RESPONSE statement gives you the opportunity to determine when the application is best able to attempt to retrieve an outstanding response.

### Conditional Clauses (WHEN)

The optional set of WHEN clauses on the GET ASYNC RESPONSE statement, give you processing control that is not available with a synchronous cooperative flow. For example, you may not want to interrupt the application's current processing if an error is detected during the handling of an asynchronous request. Compare this with the processing of a synchronous cooperative flow, where errors are handled by the runtime servicing the flow, and the processing of the initiating PStep is aborted if the runtime detects an error in either the handling of the request or in the actual processing of the target PStep code (that is, an XFAL).

As an application developer, you may want control of the various processing conditions that may occur when servicing an asynchronous flow. Alternatively, you may prefer that the runtime process errors in the same way as they are handled for existing synchronous flows. The behavior of the GET ASYNC RESPONSE statement can be modified, depending on which of the WHEN clauses you include in the statement.

When executing the GET ASYNC RESPONSE statement, the process flow gives control to an applicable WHEN clause, if it is present. When the block of code associated with the WHEN clause has completed, processing continues with the statement immediately after the GET ASYNC RESPONSE statement. All of the WHEN clauses are included in the code when the GET ASYNC RESPONSE statement is constructed by the PAD editor. You can delete any of the WHEN clauses, as required.

Control is passed to the associated WHEN clause under the following circumstances:

- **WHEN successful**—Control is passed to the WHEN successful block of code if the response identified by the specified ASYNC\_REQUEST view is available, and has successfully resulted in the runtime populating the specified view (specified by the WHICH EXPORTS clause). The outstanding request is considered complete as a result of successfully executing the GET ASYNC RESPONSE statement.

If the statement is not present, processing continues with the statement following the GET ASYNC statement.

- **WHEN pending**—The behavior of the GET ASYNC RESPONSE statement varies depending on whether the WHEN pending clause is present. If present, active polling is used, and the response identified by the specified ASYNC\_REQUEST Work Set view is not available, control is passed to the WHEN pending block of code. That is, the outstanding request remains outstanding, and the execution of the GET ASYNC RESPONSE statement operates as a non-blocking operation.

If the WHEN pending clause is not present, passive polling is used, and the GET ASYNC RESPONSE statement causes the application to block until the response is available. Processing is returned to the application only when the asynchronous request is completed. The appropriate WHEN clause is driven when processing the completed request.

- **WHEN invalid ASYNC\_REQUEST ID**—If the specified request identifier does not correspond to an outstanding request, processing continues at the first statement contained in the WHEN Invalid ASYNC\_REQUEST ID.

If the WHEN invalid ASYNC\_REQUEST ID is not present, processing continues at the default error handling code generated for this statement. Processing exits the currently executing code. This may be handled differently if within a GUI event or within an action block.

**Note:** For more information, see the *Distributed Processing – Overview Guide*.

- **WHEN server error**—If this clause is present, and the response returned from the server contains a server failure (that is, an XFAL buffer), control passes to the first statement contained within the WHEN server error clause.

If the WHEN server error clause is not present, the CA Gen runtime continues at the default error handling code generated for this statement. That is, the currently executing code is exited. This may be handled differently if within a GUI event or within an action block.

**Note:** For more information, see the *Distributed Processing – Overview Guide*.

- **WHEN communications error**—If this clause is present, and the processing of the asynchronous flow encounters a problem in either the transport of the request or receipt of the response (that is, an XERR buffer), then processing is passed to the first statement contained within the WHEN communications error clause.

If this clause is not present, control is passed to the default error handling code generated for this statement. This default code causes the processing to exit the currently executing code. This may be handled differently if within a GUI event or within an action block.

**Note:** For more information, see the *Distributed Processing – Overview Guide*.

## CHECK ASYNC RESPONSE

This command description includes the following information:

- The syntax of the command
- A description of the parameters

### Syntax

```
+--CHECK ASYNC RESPONSE
| IDENTIFIED BY: viewname async_request
+--WHEN available
+--WHEN pending
+--WHEN invalid ASYNC_REQUEST_ID
```

An application can check the state of a given outstanding asynchronous request, without actually obtaining its associated response, by using the CHECK ASYNC RESPONSE statement. This statement executes as a non-blocking operation. The runtime interrogates the state of the specified request, then communicates that state to the application by passing control to one of the specified blocks of code that has been defined in the associated WHEN clauses of the statement.

If the application just needs to check the status of a response, using CHECK ASYNC RESPONSE is more efficient, and faster, than using GET ASYNC RESPONSE as it does not need to allocate (then de-allocate) any resources (for example, storage for the response message object).

## Request Identifier (IDENTIFIED BY:)

The application indicates which specific request is to be checked by the ASYNC\_REQUEST work set view, as part of the IDENTIFIED BY clause. The specified ASYNC\_REQUEST work set view that is defined should be the same view (or at least a view that contains the same values) as was specified on the USE ASYNC statement that initiated the original asynchronous flow.

## Conditional Clauses (WHEN)

When executing the CHECK ASYNC RESPONSE statement, the process flow gives control to an applicable WHEN clause, if it is present. When the block of code associated with the WHEN clause has completed, processing continues with the statement immediately after the CHECK ASYNC RESPONSE statement. All of the WHEN clauses are included in the code when the CHECK ASYNC RESPONSE statement is constructed by the PAD editor. You can delete any of the WHEN clauses, as required.

Control is passed to the associated WHEN clause under the following circumstances:

- **WHEN available**—Control is passed to the WHEN available block of code if the response associated with the specified ASYNC\_REQUEST Work set view has been received by the runtime, and is available for the application to retrieve via a GET ASYNC RESPONSE statement. Note that the WHEN available clause does not imply the success or failure of the actual request, it only indicates that the response is available to be retrieved by the application.

If this clause is not present, there is no way to know if the response is available using the CHECK ASYNC RESPONSE. Processing continues at the statement following the CHECK ASYNC RESPONSE statement, as in the case where the response is still pending.

- **WHEN pending**—If this clause is present, and the identified request has not yet completed execution, then control is passed to the WHEN pending block of code. The associated request is still considered outstanding.

If this clause is not present, there is no way to know if the response is available using the CHECK ASYNC RESPONSE. Processing continues at the statement following the CHECK ASYNC RESPONSE statement, the same as if the response is still pending.

- **WHEN invalid ASYNC\_REQUEST ID**—If the specified request identifier does not correspond to an outstanding request, control is passed to the first statement contained in the WHEN invalid ASYNC\_REQUEST ID.

If the WHEN invalid ASYNC\_REQUEST ID is not present, processing continues at the default error handling code generated for this statement. Processing exits the currently executing code. This may be handled differently if within a GUI event or within an action block. For more information, see the appendix "User Exits" in the *Distributed Processing – Overview Guide*.

## IGNORE ASYNC

This command description includes the following information:

- The syntax of the command
- A description of the parameters

### Syntax

```
+--IGNORE ASYNC RESPONSE  
| IDENTIFIED BY: viewname async_request  
+--WHEN invalid ASYNC_REQUEST_ID
```

It is likely that an application will, at some point, have one or more outstanding asynchronous requests that it no longer wishes to process. You can use the IGNORE ASYNC RESPONSE statement to inform the runtime that the application wishes to ignore the response corresponding to the specified ASYNC\_REQUEST.

However, the runtime will continue to keep track of an ignored request, as the resources associated with an ignored request remain allocated as long as the corresponding request of the ignored request remains outstanding. Only when the response is received (that is, the request is complete), can the runtime free the resources associated with the request.

An ignored outstanding request is considered complete.

### Request Identifier (IDENTIFIED BY:)

The application identifies the specific request that is to be ignored by the ASYNC\_REQUEST view, specified in the IDENTIFIED BY clause. The specified ASYNC\_REQUEST view should be the same view (or at least a view that contains the same values) as was specified on the USE ASYNC statement that initiated the given asynchronous flow.

### Conditional Clause (WHEN)

Control is passed to the associated WHEN clause under the following circumstance:

**WHEN invalid ASYNC\_REQUEST ID**—If the specified request identifier does not correspond to an outstanding request, control is passed to the first statement contained in the WHEN Invalid ASYNC\_REQUEST ID.





# Index

---

## A

- Accessing • 127, 151, 152, 154, 158
  - Action Block Usage tool • 152
  - Procedure synthesis • 158
  - Process synthesis • 127
  - Structure Chart tool • 151
  - View maintenance • 154
- Action • 14, 31, 59, 64, 73, 74, 75, 76, 77, 83, 105, 118, 119, 120, 131, 154
  - Adding a blank line • 77
  - Adding conditional • 59
  - Adding control • 73
  - Adding entity • 14
  - Adding event • 83
  - Adding miscellaneous • 76
  - Detail entity action views • 105
  - ESCAPE • 75
  - Identify required • 131
  - Identifying sequence • 131
  - MAKE • 76
  - NEXT • 74
  - Procedure Step USE • 74
  - Relationship • 31
  - Repeating • 64
  - SET USING assignment • 120
  - Updating USE control • 119
  - USE • 74
  - USE control action • 118
  - Using other options • 154
- Action bar for command entry, adding • 162
- Action Block • 78, 79, 80, 153, 154
  - Accessing view maintenance • 154
  - Adding • 153
  - Changing the name • 153
  - Creating • 78
  - Defining • 78
  - Defining external Action Blocks • 80
  - Defining for processes and procedures • 78
  - Describing • 154
  - Detailing • 153
  - Reference common • 79
- Action Block Usage tool • 151, 152, 153
  - Accessing • 152
  - Using • 153

- Action Diagram • 9, 10, 11, 12, 77, 80, 81, 138, 141
  - Adding NOTE • 77
  - Building • 11
  - Changing • 80
  - Complex changes • 80
  - Copying • 81
  - Procedure • 10
  - Process • 9
  - Process synthesis • 138
  - Reviewing generated • 141
  - Simple changes • 81
  - Using view maintenance • 12
- Activity Model, elementary processes • 13
- Adding a nested repeating group view • 96
- Adding an Action Block • 153
- Adding attributes to a work set • 100
- Adding views • 93
- Assigning view characteristics • 105
- Associate views with external objects (add) • 113
- Attribute • 100, 101
  - Adding to a work set • 100
  - Detailing in work set • 101

## B

- Building the Action Diagram • 11

## C

- CA Gen-supplied event types • 84
- Changing • 80, 109, 110, 153
  - Action Diagram • 80
  - Name of action block • 153
  - Parent of views • 110
  - Views • 109
- Clearing data from screen • 167
- Complex changes • 80
- Conditional actions, adding • 59
- Conditions • 60, 62, 63, 69
  - Complex • 62
  - Dependent on attribute values • 63
  - FOR EACH • 69
  - Repeat • 69
  - Simple • 60
- Control actions, adding • 73
- Copying • 81, 110, 120
  - Action Diagram • 81

---

- And matching views • 120
- Views • 110
- Creating a Procedure Action Diagram • 157
- Creating a view • 109
  - Recommendations • 109
- Creating new action block in Analysis or Design • 78
- Creating process logic diagram • 129

## D

- Defining Action Blocks • 78
- Defining logic for elementary processes and procedures • 13
- Deleting views • 110
- Describing an action block • 154
- Detailing an action block • 153
- Detailing attributes in work set • 101
- Detailing entity action views • 105
- Detailing import • 96, 98, 100
  - Entity views • 98
  - Group views • 96
  - Work views • 100
- Display using Enter command • 166
- DLG, synthesize views • 113
- Dot notation • 87

## E

- Elementary process • 13, 14, 130
  - Defining logic • 13
  - Generate • 14
  - In Activity Model • 13
  - Selecting to analyze • 130
- Enter command • 166
  - Using to display • 166
  - Using to update • 166
- Entity • 97, 98, 105, 160
  - Adding import entity views • 98
  - Detail entity action views • 105
  - Detail import entity views • 98
  - Import entity views • 97
  - Maintenance stereotype • 160
- Entity action • 14, 104, 105
  - Adding • 14
  - Adding views • 104
  - Creating views in view subset • 104
  - Supports option • 105
  - Views • 104
- Entry, lock required • 107
- ESCAPE action, adding • 75

- Event action • 83
  - Adding • 83
  - Names • 83
- Event processing • 83
- Event types • 84, 85
  - CA Gen-supplied • 84
  - User-defined • 85
- Executing entity actions • 164
- Export view subset, creating views • 101
- External Action Blocks, defining • 80
- External objects, associate views (add) • 113

## F

- FOR EACH condition, adding • 69

## G

- Generated • 139, 140, 141, 142, 144, 146, 147
  - Action diagrams • 141
  - Definition properties • 139
  - Expected effects • 140
  - PAD for CREATE • 142
  - PAD for DELETE • 146
  - PAD for LIST • 147
  - PAD for READ • 144
  - Processes • 139
- Generating elementary processes • 14
- GUI object domain • 87
- GUI statements • 86
  - Action Diagram statements • 86
- Guidelines for matching views • 116

## I

- Identify entity types used in process • 130
- Implementing action blocks • 161
- Import entity view • 97, 98, 99
  - Adding • 98
  - Creating • 97
  - Specifying optionality • 99
- Import group view • 94, 96, 97
  - Adding • 94
  - Creating • 94
  - Specifying cardinality • 96
  - Specifying optionality • 97
- Import view subset, creating views • 94
- Import work • 99, 100
  - Set description • 100
  - Views • 99

---

## L

Local view subset, creating views • 103  
Lock required on entry • 107

## M

MAKE action, adding • 76  
Matching and copying views • 120  
Matching views • 115, 116, 117  
    Guidelines • 116  
    Using the PAD • 117  
Maximum size of views • 121  
Miscellaneous actions, adding • 76

## N

Nested repeating group view • 96  
NEXT action, adding • 74  
NEXTLOCATION system attribute, adding • 42  
NOTE, adding • 77

## P

PAD • 117, 142, 144, 145, 146, 147  
    Reviewing generated PAD for CREATE • 142  
    Reviewing generated PAD for DELETE • 146  
    Reviewing generated PAD for LIST • 147  
    Reviewing generated PAD for READ • 144  
    Reviewing generated PAD for UPDATE • 145  
    Using to match views • 117  
Parent of views • 110  
Performing simple changes • 81  
Prerequisites, Procedure Synthesis • 158  
Procedure action diagram • 10, 157  
    Creating • 157  
Procedure definition • 157  
Procedure step USE actions, adding • 74  
Procedure synthesis • 157, 158, 159, 161, 162, 164, 166, 167, 168, 169, 170  
    Accessing • 158  
    Adding an action bar for command entry • 162  
    Clearing data from screen • 167  
    Creating procedure action diagram • 157  
    Executing entity actions • 164  
    Implementing action blocks stereotype • 161  
    Prerequisites • 158  
    Procedure definition • 157  
    Refreshing screen on cancel • 166  
    Resynthesize processes • 170  
    Selecting stereotype • 159

    Selecting stereotype options • 162  
    Selection list stereotype • 161  
    Synthesize flows • 168  
    Synthesize menu flows • 168  
    Synthesize neighbor selection list flows • 169  
    Synthesize subject list flows • 169  
    Tasks • 159  
    Using • 159  
    Using Enter command to display • 166  
    Using Enter command to update • 166  
Process action diagrams • 9  
Process logic diagram • 129, 130  
    Creating • 129  
    Selecting elementary process to analyze • 130  
Process synthesis • 123, 124, 126, 127, 129, 130, 131, 132, 133, 134, 135, 136, 138, 139  
    Accessing • 127  
    Creating process logic diagram • 129  
    From Action Diagram tool • 138  
    From Activity Hierarchy and Activity Dependency tools • 138  
    From Data Model • 136  
    Identifying entity types • 130  
    Identifying required actions • 131  
    Identifying sequence of actions • 131  
    Output • 124  
    Performing • 132  
    Prerequisites • 126  
    Reviewing results • 139  
    Rules • 133  
    Rules for CREATE • 134  
    Rules for DELETE • 136  
    Rules for UPDATE • 135  
    Using • 129  
    When to use • 123

## R

REASON\_CODE • 171  
Recommendations for creating a view • 109  
Refresh screen on cancel • 166  
Relationship actions, adding • 31  
Repeat conditions, adding • 69  
Repeating actions, adding • 64  
Reviewing results of process synthesis • 139, 140, 141, 142, 144, 146, 147  
    Generated action diagrams • 141  
    Generated definition properties • 139  
    Generated expected effects • 140

---

- Generated PAD for CREATE • 142
- Generated PAD for DELETE • 146
- Generated PAD for LIST • 147
- Generated PAD for READ • 144
- Generated processes • 139
- Generated views • 140
- Rules • 134, 135, 136
  - CREATE • 134
  - DELETE • 136
  - UPDATE • 135

## S

- Selecting elementary process to analyze • 130
- Selection list stereotype • 161
- SET USING assignment • 120
- Simple conditions, adding • 60
- Size maximum of views • 121
- Stereotype • 159, 160, 161, 162
  - Entity maintenance • 160
  - Implementing action blocks • 161
  - Selecting • 159
  - Selecting options • 162
- Structure Chart tool • 150, 151, 153
  - Accessing • 151
  - Using • 153
- Supports entity action option • 105
- Synthesize • 113, 168, 169
  - Flows • 168
  - Menu flows • 168
  - Neighbor selection list flows • 169
  - Subject list flows • 169
  - Views in the DLG • 113
- System attribute • 42
  - Adding NEXTLOCATION • 42

## U

- Unmatching views • 120
- Updating USE control action • 119
- Updating using Enter command • 166
- USE action, adding • 74
- USE control action • 118, 119
  - Adding • 118
  - Updating • 119
- USE Procedure Step • 171
  - ASYNC\_REQUEST • 171
- Used as both Input and Output option • 108
- User-defined event types • 85
- Using Action Diagram view maintenance • 12

- Using Enter command • 166
  - To display • 166
  - To update • 166
- Using other action options • 153, 154, 155, 156
  - Chain • 154
  - Check • 155
  - Contract • 155
  - Expand • 155
  - Expand All • 155
  - Matching views • 154
  - Redraw • 155
  - Unmatch • 156
  - Using diagram • 153
- Using process synthesis • 129
- Using views • 113

## V

- View • 11, 12, 89, 90, 93, 94, 96, 97, 98, 99, 100, 101, 103, 104, 105, 108, 109, 110, 111, 113, 115, 116, 117, 120, 121, 140, 154
  - Adding • 93
  - Adding a nested repeating group • 96
  - Adding import entity • 98
  - Adding import work • 99
  - Assigning characteristics • 105
  - Associate with external objects (add) • 113
  - Change parent • 110
  - Changing • 109
  - Copying • 110
  - Creating • 11
  - Creating import entity • 97
  - Creating import group • 94
  - Creating import work • 99
  - Creating in entity action subset • 104
  - Creating in export view subset • 101
  - Creating in import view subset • 94
  - Creating in local view subset • 103
  - Deleting views • 110
  - Describing import entity • 99
  - Describing import group • 96
  - Detailing entity action views • 105
  - Detailing import entity views • 98
  - Detailing import group views • 96
  - Detailing import work • 100
  - Entity action • 104
  - Guidelines for matching • 116
  - Import entity view optionality • 99
  - Import group • 94

---

- Import group view cardinality • 96
- Import group view optionality • 97
- Introduction to view • 89
- Maintenance • 154
- Matching • 115
- Matching and copying • 120
- Matching using the PAD • 117
- Maximum size • 121
- Moving • 111
- Of work set • 99
- Recommendations for creating • 109
- Renaming • 111
- Reviewing generated • 140
- Synthesize in the DLG • 113
- Unmatching • 120
- Used as both Input and Output option • 108
- Using • 90
- Using Action Diagram view maintenance • 12
- Using view maintenance • 12

## W

- When to use process synthesis • 123
- Work attribute set • 99
- Work set • 99, 100, 101, 171
  - Adding attributes • 100
  - Describing import • 100
  - Detailing attributes • 101
  - View • 99