

Using Application Engine

Application Engine (A/E), a PeopleTool introduced in version 5.0, offers an alternative to conventional structured programming. Application Engine programs are created using a series of online panels which allow you to define your application along with any section, step, and statement components. You can use radio buttons, checkboxes, and drop-down lists to designate execution options in your program. SQL statements are entered on the statement panel in an edit box. All of this information is saved within the database and utilized by the Application Engine driver program PTPMAIN. Because Application Engine is not an intuitive development tool, we follow our introduction to Application Engine with a “hands-on” approach, presenting a series of exercises in a tutorial designed to illuminate the differences between A/E and conventional structured programming. As a prerequisite to part 7, the user should be well-versed in PeopleTools, particularly Application Designer. A good understanding of SQR programming (as well as SQL) would also be helpful. We end the section with an overview of new Application Engine features in release 8.0.



CHAPTER 35

What is Application Engine?

35.1 About Application Engine	771	35.5 A/E definition tables	777
35.2 Advantages/disadvantages	772	35.6 A/E definition panels	779
35.3 Set processing concepts	772	35.7 A/E section/step relationship	786
35.4 The main components of Application Engine	776	35.8 Application Engine: the big picture	788

35.1 ABOUT APPLICATION ENGINE

Application Engine (A/E) is a PeopleTool that allows you to create and execute Batch SQL programs. SQL statements are entered online and processed by the PeopleSoft COBOL program PTPMAIN. Applications can be broken into smaller pieces called Sections and within these Sections are Steps. Each Step either executes SQL statements, another Section, a COBOL program, or a Mass Change program. In structured programming languages such as SQR, variables are used to store information throughout the life of the program. In Application Engine, a Cache record is used to store values so they may be utilized by subsequent steps in the AE program. As your program proceeds through its Sections/Steps, messages may be written, which are stored in the Message Log tables. These messages may be viewed through the Application Engine Messages panel.

35.2 ADVANTAGES/DISADVANTAGES

Application Engine offers both advantages and disadvantages:

35.2.1 Advantages

- All Application Engine components reside within the database itself. All application development and testing is done within PeopleTools.
- Application Engine programs are considered multi-platform. Database-specific sections can be utilized using a database platform directive that matches your particular installation.
- PeopleSoft Meta-SQL is supported within Application Engine.
- Changes to the PeopleSoft data dictionary are global. No modifications to Application Engine programs are normally required when a field attribute is changed
- Application Engine programs use effective-dating for each section (or procedure). A history of modifications can be easily maintained instead of overlaid.
- Extremely efficient programs can be created using set processing techniques.

35.2.2 Disadvantages

- Application Engine panels are not intuitive: It can be confusing scrolling through a maze of checkboxes, radio buttons, and folder tabs.
- It is difficult to visualize the flow of an Application Engine program. Sections are stored and displayed in alphabetical order in the list box instead of a more logical order.
- Even the simplest of modifications to Application Engine programs can be a harrowing experience. Some more complex programs need to use temporary tables to pass information from one step to another. The dependencies on these temporary tables by other sections need to be carefully analyzed.

35.3 SET PROCESSING CONCEPTS

The most efficient Application Engine programs use set processing techniques whenever possible. In fact, Application Engine was designed with this technique in mind. Large groups of data with the same criteria can be processed at once instead of individually (or row-by-row). Depending on the volume of data processed, set processing can dramatically improve the overall performance of your program.

The set processing SQL concept has been around for many years. It is used extensively when updating the database using native SQL tools such as SQL*Plus or SQL*Talk. Set processing can also be referred to as a mass update. These mass updates may be split into several SQL statements to accommodate different sets of update criteria for each group of data. Let's look at a simplified example of set processing before we move on to Application Engine Basics. We'll use basic SQR routines to demonstrate set processing in comparison to row by row processing.

35.3.1 Set processing vs. row by row processing

For our example, let's assume we have a record called MY_TABLE. Many fields exist in the table including MY_KEY, DEPTID, and ACCT_TYPE. MY_KEY will serve as the unique table key. Also included is a field called BUSINESS_UNIT, which is not populated. Based on the ACCT_TYPE value, we need to perform two different methods of deriving the BUSINESS_UNIT using the DEPTID field.

If the ACCT_TYPE has a value of 'A', BUSINESS_UNIT will be extracted from a table called MY_CONV_A. If ACCT_TYPE has a value of 'B', the BUSINESS_UNIT will be extracted from a table called MY_CONV_B.

35.3.2 Example of row by row processing

First, we'll use the row-by-row processing technique to derive the business unit:

```
...  
  
begin-select  
  
u.my_key  
u.deptid  
u.acct_type  
  
    let $NEW_business_unit = ''  
  
    if &u.acct_type = 'A'  
        do Select-Conv-A  
    else  
        do Select-Conv-B  
    end-if  
  
    if not isnull($NEW_business_unit)  
        do Update-My-Table  
    end-if  
  
from ps_my_table  
where u.acct_type in ('A','B')  
  
end-select  
  
...
```

The main Select as indicated, fetches a row from MY_TABLE one by one. Only rows with ACCT_TYPE of 'A' or 'B' are selected. If ACCT_TYPE is equal to 'A', then the Select-Conv-A routine is performed. If ACCT_TYPE is not equal to 'A', then the Select-Conv-B routine is performed by default. If a BUSINESS_UNIT is found in either of these tables, then the routine Update-My-Table is performed. This process will continue until all rows with ACCT_TYPE equal to 'A' or 'B' have been processed.

Let's look at the `Select-Conv-A` routine. If there is a matching entry in the `MY_CONV_A` table for the `DEPTID`, then the `$NEW_business_unit` variable will be set to the `BUSINESS_UNIT` value in the table:

```
begin-procedure Select-Conv-A

begin-select

a.business_unit

    let $NEW_business_unit = &a.business_unit

    from ps_my_conv_a  a
    where a.deptid     = &u.deptid

end-select

end-procedure
```

Let's look at the `Select-Conv-B` routine. If there is a matching entry in the `MY_CONV_B` table for the `DEPTID`, then the `$NEW_business_unit` variable will be set to the `BUSINESS_UNIT` value in the table:

```
begin-procedure Select-Conv-B

begin-select

b.business_unit

    let $NEW_business_unit = &b.business_unit

    from ps_my_conv_b  b
    where b.deptid     = &u.deptid

end-select

end-procedure
```

Finally, let's have a look at the routine that updates `MY_TABLE` with the new `BUSINESS_UNIT` value (if a matching entry were found):

```
begin-procedure Update-My-Table

begin-sql

update ps_my_table
    set business_unit = $NEW_business_unit
    where my_key      = &u.my_key

end-sql

end-procedure
```

Depending on the volume of data that will be processed, the row by row approach may be fine. At a minimum, each row selected from MY_TABLE must perform a Select to retrieve the Business Unit. If it exists, an Update statement is performed. Imagine if the number of rows affected by this process were over 100,000. Maybe even 500,000 or more! This means the Select against a conversion table would be executed that many times. The Update routine could potentially execute the same amount of times! That's a lot of database activity that could be avoided! Network traffic, which could yield the greatest degradation in performance, needs to be considered.

We can implement optimization techniques in our row by row processing example to improve performance. For example, we can order the main Select by ACCT_TYPE and DEPTID. We then perform a Conversion Lookup only when a change to one of these fields occurs. We store and utilize the results based on the changing combination of these two fields in our update routine. Even with these improvements, performance can still be poor when processing large amounts of data one row at a time.

35.3.3 Example of set processing

Our set processing example is much simpler and much more efficient. The improvement in performance increases with the volume of transactions processed. Let's perform the set processing Update using the MY_CONV_A table:

```
...
begin-sql

update ps_my_table          u
   set u.business_unit      =
      (select z.business_unit
        from ps_my_conv_a    z
       where z.deptid        = u.deptid)
 where u.acct_type          = 'A'
    and exists
      (select 'X'
        from ps_my_conv_a    x
       where x.deptid        = u.deptid);

commit;

end-sql

...
```

The WHERE clause limits the Update to all rows in MY_TABLE that have ACCT_TYPE equal to 'A' and an existing entry in the MY_CONV_A table equal to the DEPTID on the row. This one Update statement populates all the rows that

match this criteria at once. Now let's perform the set processing `Update` using the `MY_CONV_B` table:

```
...  
  
begin-sql  
  
update ps_my_table          u  
  set u.business_unit      =  
      (select z.business_unit  
        from ps_my_conv_b   z  
        where z.deptid      = u.deptid)  
where u.acct_type          = 'B'  
  and exists  
      (select 'X'  
        from ps_my_conv_b   x  
        where x.deptid      = u.deptid);  
  
commit;  
  
end-sql  
  
...
```

The `WHERE` clause limits the `Update` to all rows in `MY_TABLE` that have an `ACCT_TYPE` equal to 'B' and an existing entry in the `MY_CONV_B` table equal to the `DEPTID` on the row. This one update statement also populates all the rows that match this particular criteria at once.

TIP It's a good idea to `COMMIT` frequently during set processing operations. Large amounts of data may be updated at once, causing more system resources to be utilized.

The set processing examples executed two SQL `Updates`. No further database activity was required, and there was no network traffic at all. The `Updates` were entirely at the database level.

Always keep set processing in mind when you're developing with Application Engine, and try to use it whenever possible to achieve the maximum performance in your programs.

35.4 THE MAIN COMPONENTS OF APPLICATION ENGINE

Application Engine contains the following main components:

APPLICATION The highest level of an Application Engine program comprised of one or more sections.

SECTION Equivalent to an SQR procedure or COBOL paragraph comprised of one or more steps. An Application Engine program always begins with a section called MAIN.

STEP Can be considered the actual work component of an Application Engine program. In most cases, it is used to execute an SQL statement or call another section. It can also call a COBOL program or a Mass Change program.

STATEMENTS An SQL statement attached to a step. Several statement types are used to qualify a statement: Select, Update/Insert/Delete, DO Select, DO When, DO Until, DO While, and Comment. The Update/Insert/Delete statement type is used not only to update the database but also to insert messages into the message log.

35.5 A/E DEFINITION TABLES

All Application Engine development is done within the database itself. Just as a record or panel definition is created and stored in the database, the same can be said of Application Engine. Using the Application Engine panels, the application is defined and stored in an application table. Next a section is defined and stored in the section table. The same occurs for each step and each statement. Four definition tables are used to store Application Engine programs along with the relationship to one another (table 35.1 through table 35.5).

Table 35.1

AE_APPL_TBL	AE_SECTION_TBL	AE_STEP_TBL	AE_STMT_TBL
AE_PRODUCT	AE_PRODUCT	AE_PRODUCT	AE_PRODUCT
AE_APPL_ID	AE_APPL_ID	AE_APPL_ID	AE_APPL_ID
	AE_SECTION	AE_SECTION	AE_SECTION
	DB_PLATFORM	DB_PLATFORM	DB_PLATFORM
	EFFDT	EFFDT	EFFDT
		AE_STEP	AE_STEP
			AE_STMT_TYPE

We will now briefly describe some of the more important fields stored in each of these tables.

Table 35.2

AE_APPL_TBL	Application Definition Table
AE_PRODUCT	Product
AE_APPL_ID	Application Name
AE_VERSION	Version Number

Table 35.2 (continued)

AE_APPL_TBL	Application Definition Table
DESCR	Description
AE_CACHE_RECNAME	Cache Record Name
MESSAGE_SET_NBR	Message Set Number
AE_DEBUG_MODE	Debug Application
AE_TRACE	Trace Application Steps

Table 35.3

AE_SECTION_TBL	Section Definition Table
AE_PRODUCT	Product
AE_APPL_ID	Application Name
AE_SECTION	Section
DB_PLATFORM	Database Platform
EFFDT	Effective Date
EFF_STATUS	Effective Status
DESCR	Description

Table 35.4

AE_STEP_TBL	Step Definition Table
AE_PRODUCT	Product
AE_APPL_ID	Application Name
AE_SECTION	Section
DB_PLATFORM	Database Platform
EFFDT	Effective Date
AE_STEP	Step Name
AE_SEQ_NUM	Step Sequence Number
EFF_STATUS	Effective Status
PROGRAM_NAME	COBOL Program
MC_DEFN_ID	Mass Change Definition
AE_DO_PRODUCT	DO Product
AE_DO_APPLID	DO Application
AE_DO_SECTION	DO Section
AE_SQL_UPDATE	Edit SQL
AE_SQL_SELECT	Select Present
AE_DO_WHEN	When
AE_DO_WHILE	While
AE_DO_UNTIL	Until

Table 35.4 (continued)

AE_STEP_TBL	Step Definition Table
AE_DO_SELECT	Select
AE_SELECT_END_DO	Select Ends the DO
AE_DO_SELECT_TYPE	Type of DO Select

Table 35.5

AE_STMT_TBL	Statement Definition Table
AE_PRODUCT	Product
AE_APPL_ID	Application Name
AE_SECTION	Section
DB_PLATFORM	Database Platform
EFFDT	Effective Date
AE_STEP	Step Name
AE_STMT_TYPE	Statement Type
AE_STMT	SQL Statement

Be aware that an additional table, AE_STMT_B_TBL, which is a statement chunk, table exists. This is used to store the SQL statements entered in AE_STMT_TBL into smaller pieces or chunks once you save the definitions using the online panels. When an Application Engine program is executed, the chunks are selected and pieced together to form the original statement entered. This alleviates any incompatibility problems using Long datatypes in other databases. The synchronization between the AE_STMT_TBL and AE_STMT_B_TBL may become corrupted. An option does exist on the Application Definition panel which rebuilds the chunked statements. We'll identify this option in the pages ahead. Keep in mind that the breakdown of the SQL statements is done in the background.

As you begin constructing your Application Engine program through the online panels, the fields in each of these tables will be populated based on the selections you make. For example, a statement type of DO when sets the AE_DO_WHEN indicator in the AE_STEP_TBL. You have to fill in the name of your application, sections, and steps along with the descriptions. The statements themselves must also be filled in manually. Most of the remaining options are selected using radio buttons, drop-down lists, and checkboxes.

35.6 A/E DEFINITION PANELS

The Application Engine Definitions for each application, section, step, and statement are entered through a series of panels. You can navigate freely through these panels

using the folder tabs at the top or through some strategically placed push buttons. Let's take a look at the panels for each of the A/E categories (figures 35.1-35.4).

35.6.1 Application definition panel

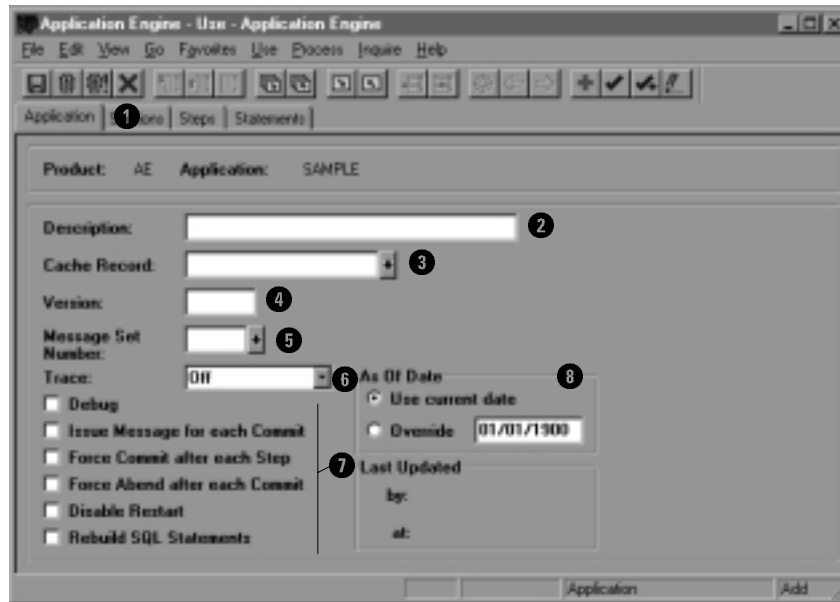


Figure 35.1 Application definition panel

- ❶ application folder tab, navigates through the four A/E panels
- ❷ description of application
- ❸ cache record used by application to store and pass values from one step to another
- ❹ version number (information only)
- ❺ default message set number, writes messages as our program progresses
- ❻ Trace options are used to create a trace file. Options are
 - Off* NO trace file produced
 - Steps Only* Each executed step is displayed on trace file showing time, section, step, and statement type.
 - SQL* In addition to Steps Only, the executed SQL is displayed on the trace file.
 - Abend Trap* Same information on trace file as SQL option. The trace file output is appended to any prior output for the same run rather than creating a new version. This creates a historical trace file that shows when an application prematurely aborted and was restarted.

- 7 Processing checkboxes are used to control the behavior of Application Engine.

Debug puts the application in interactive debugging mode. This allows you to set breakpoints, view the cache record contents, execute one step at a time, and issue commits and rollbacks.

Issue Message for each commit writes a message for each executed commit. It is recommended to use the trace option instead due to the volume of messages that may be produced by this feature.

Force Commit after each step instructs Application Engine to commit each step as the default method. Each section and step can override this if need be.

Force Abend after each commit is used for testing purposes. This is used to test your application restart capability. You can continually execute and restart the application until it is completed. If this cannot be done successfully, the program will need to be corrected.

Disable Restart allows you to restart your application from the beginning even if an abend occurs. Under normal circumstances a restart would be required. Use caution when using this option.

Rebuild SQL Statements is used to repopulate the Statement Chunk table (AE_STMT_B_TBL). When you create an SQL statement, it is stored in chunks. If you believe your chunked statements are out of sync with the statements entered, you can use this option to rebuild them.

- 8 The default date is used when filling in the effective date for each section. Sections are effective-dated. If development is spread out over several days, it's convenient to have the same effective date used when creating each new section.

35.6.2 Section definition panel

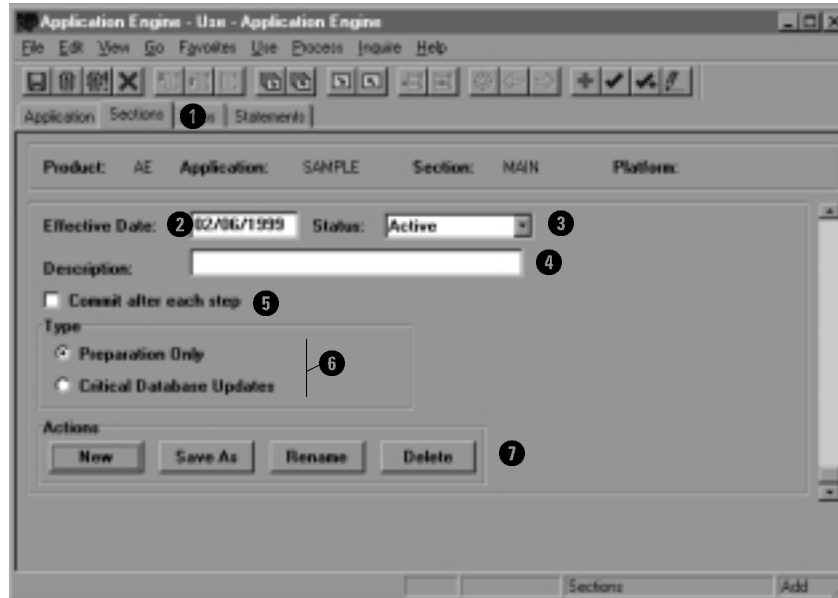


Figure 35.2 Section definition panel

- ❶ section folder tab, navigates through the four A/E panels
- ❷ effective date of the section
- ❸ effective status of the section
- ❹ description of application
- ❺ commit after each step within the section (This overrides the default setting.)
- ❻ Type option, used to designate the type of update being performed by the section

Critical Database Update should be used when a section could affect the integrity of the database in the event of an abend. A restart would be mandatory under these circumstances. Application Engine uses this indicator to update a column called `AE_CRITICAL_PHASE` in the `AE_RUN_CONTROL` table. This column will be set to 'Y' and can be used to determine if a restart is necessary.

Preparation Only simply means the section does not perform any critical database updates.

NOTE When trying to determine if a restart is necessary you can't rely totally on the AE_CRITICAL_PHASE indicator. If it is set to 'Y', you should definitely restart. If it does not equal 'Y', you may still need to restart your application. A prior step may have had critical database updates that need to be propagated in subsequent steps not yet executed. Extreme caution should be used to prevent integrity problems.

- 7 The action buttons manage your section development:
 - New* adds a new section.
 - Save As* can be used to copy the current section to a new name.
 - Rename* renames the current section.
 - Delete* deletes the current section.

35.6.3 Step definition panel

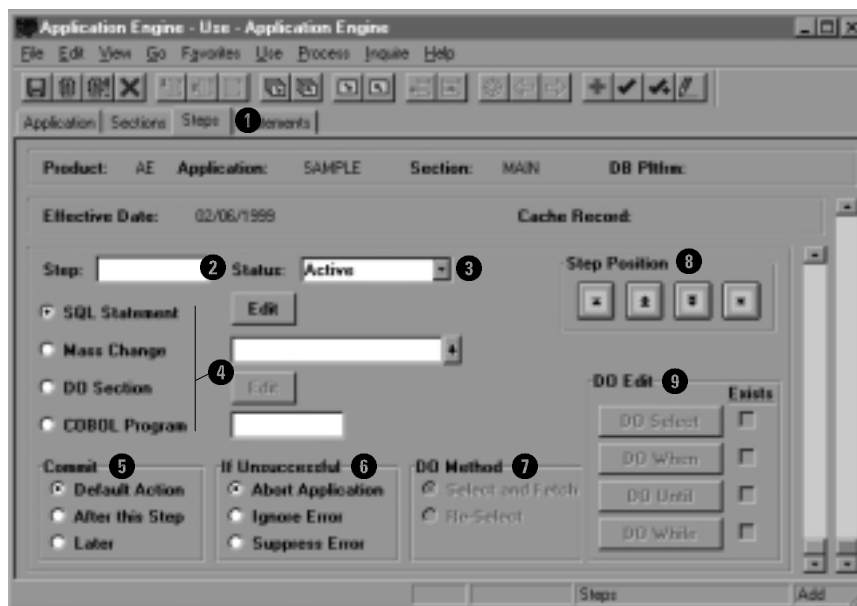


Figure 35.3 Step definition panel

- 1 step folder tab, navigates through the four A/E panels
- 2 the name of your step
- 3 effective status of the step

4 type of step

SQL Statement is used to perform an SQL or Application Engine statement entered in the Statements Panel. To access the Statements panel, press the SQL Statement Edit push button OR click on the Statement folder tab.

Mass Change allows you to execute a Mass Change program. The Mass Change definition ID can be selected from the drop-down list.

DO Section allows you to call another section. The called section can exist in your current application or an entirely different application. Sections can also be called dynamically at run time. Click on the DO Section Edit push button to access the DO Section properties dialog box. Here you will enter the DO section attributes.

COBOL Program allows a COBOL program to be called.

5 Commit override attributes for the current step

6 Error handling instructions

Abort Application performs a rollback and stops the process.

Ignore Error writes a message log entry and continues processing.

Suppress Error continues processing without any messages.

7 DO Method for DO Select statement types

The *Select and Fetch* method executes the DO Select statement once and fetches the rows one at a time. For each row fetched, the DO section is executed until all the rows have been processed.

The *Re-Select* method executes the DO Select statement and processes the first row fetched. After the first row is fetched, the DO section is executed. Upon returning, the DO Select statement is executed again, and if another row is returned, the DO section is executed again. This process continues until no rows are remaining.

8 step position buttons allow you to rearrange the order of steps in the section

9 alternate method to access DO Select statement types on the Statements panel

35.6.4 Statement definition panel

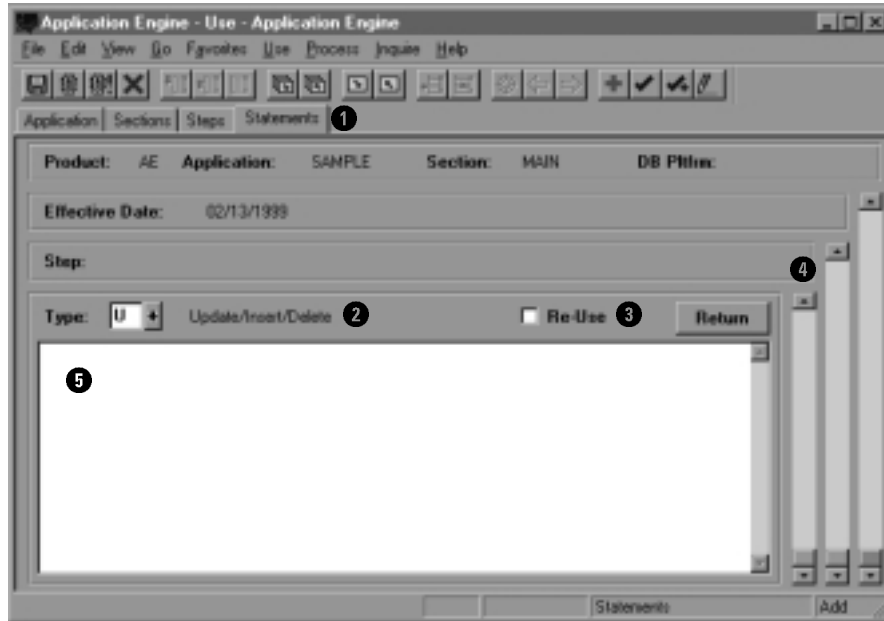


Figure 35.4 Statement definition panel

- 1 Statement folder tab, navigates through the four A/E panels
- 2 The type of statement to perform:

Comments is used to enter information about the step or to temporarily deactivate an executable statement.

Select is used to extract information and load it into your cache record.

Update statement types are used when executing SQL Updates, Inserts, or Deletes. It is also used when using the &MSG statement to write to the message log.

DO Select unconditionally executes a DO section for each row returned by the select SQL statement.

DO When conditionally executes a DO section. A *Select* statement is entered which will either return a row (representing a TRUE condition) or no rows (representing a FALSE condition). The DO section is executed when the condition is TRUE.

DO Until is used to break out of a *DO Select*. The DO section is performed and then the *DO Until* condition is evaluated. If a row is returned by the *DO Until Select* statement, the DO section is no longer performed. If no rows are returned, the DO section is repeated.

DO While is similar to an SQR or COBOL *while* function. As long as the *DO While Select* statement returns a row (representing a *TRUE* condition), the *DO* section is performed. The *DO While Select* is executed again after the *DO* section has completed. This process continues until the *DO While Select* returns a *FALSE* condition (no rows returned).

- ③ To convert *&BIND* cache fields into true bind variables, use the *Re-Use* checkbox. A true bind variable means those designated by *:1*, *:2*, etc. You may have seen these in PeopleCode's *SQLExec* functions or stored SQL statements in COBOL.

NOTE The PeopleSoft documentation does not provide a full explanation of this feature. It is used to improve performance by compiling the statement once and re-executing it with updated bind variable values. When using this feature, make sure the application is adequately tested with the desired results produced.

- ④ Click *Return* to go back to the *Step Definition* panel OR click on the *Step* folder tab.
- ⑤ Enter your SQL or Application Engine statements here.

No validation is performed on the SQL or Application Engine statement text entered. Any syntax errors will be identified at runtime. Proper testing is required for each step.

35.7 A/E SECTION/STEP RELATIONSHIP

All Application Engine programs begin with a section called *MAIN*. This can be considered the parent section when viewed in a hierarchical manner as depicted in figure 35.5. *MAIN.STEP1* executes a section called *LEVEL2A*. This level has three steps. Once all three steps have been completed, control is passed back to *MAIN*, and *MAIN.STEP2* is executed. This performs the *LEVEL2B* section. In turn, *LEVEL2B.STEP1* performs *LEVEL3*. All called sections are performed in this manner until the last step in the *MAIN* section has been completed.

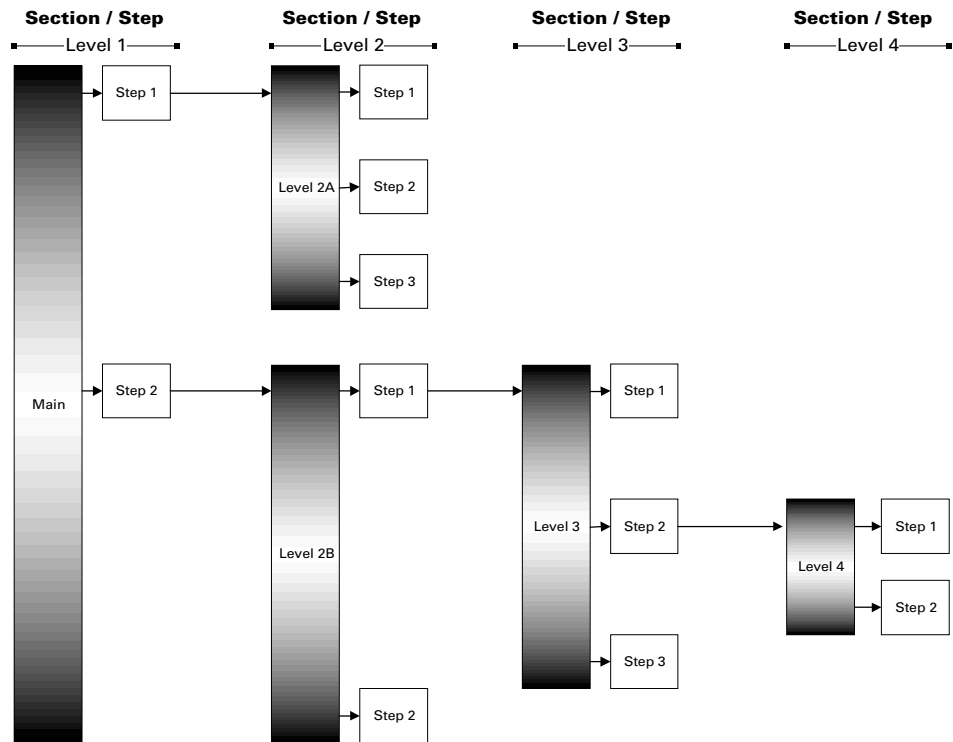


Figure 35.5 Section/step relationship

Let's take a look at the process flow (using the example in figure 35.5) as it would appear on a trace file listing. Each line represents a step executed within a section using the SECTION.STEP format:

```

MAIN.STEP1
  LEVEL2A.STEP1
  LEVEL2A.STEP2
  LEVEL2A.STEP3
MAIN.STEP2
  LEVEL2B.STEP1
    LEVEL3.STEP1
    LEVEL3.STEP2
      LEVEL4.STEP1
      LEVEL4.STEP2
    LEVEL3.STEP3
  LEVEL2B.STEP2
  
```

This is the basic execution structure of an Application Engine program. Visualizing the process in this manner will help tremendously when creating a new program or modifying an existing one.

35.8 APPLICATION ENGINE: THE BIG PICTURE

If you look at the “big picture” in figure 35.6 you will see the heart of the Application Engine is the COBOL process PTPemain. This is what controls each action being performed. When a process request is submitted, the PTPemain program is called. It reads any parameters that may be assigned and automatically updates the cache record of your application. It reads and processes the A/E definitions you have created (application, sections, steps, statements). It compiles and executes SQL statements against the PeopleSoft tables specified. It inserts messages into the message log. The PTPemain process also maintains a special Run Control record called AE_RUN_CONTROL which tracks the last committed step for restart purposes. PTPemain handles all processing of Trace Files.

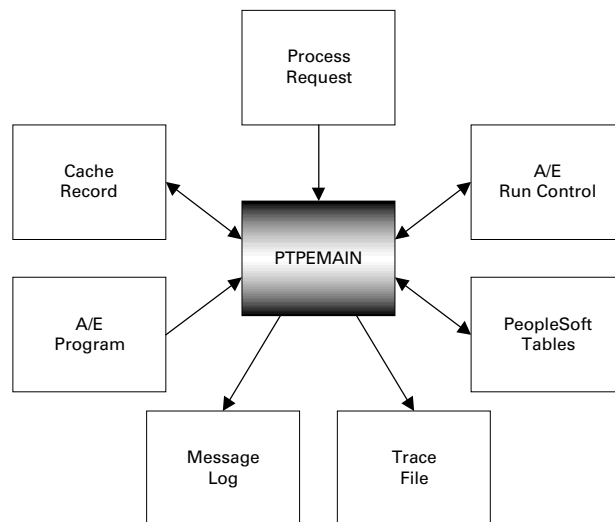


Figure 35.6 PTPemain is the “heart” of Application Engine