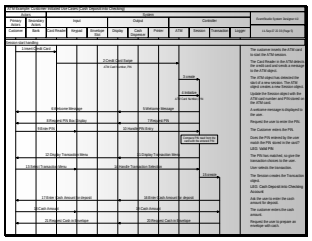
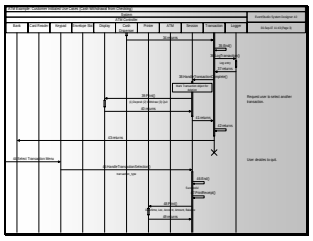
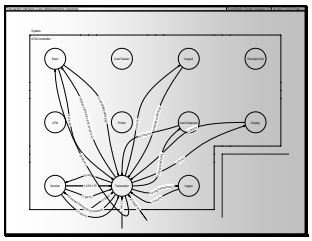
1 Use Case and Sequence Diagram Tutorial

This tutorial walks you through the development of a simple ATM from use cases to code. The following stages are covered:

Requirements	We start by defining the requirements for a simple ATM that allows the users to withdraw and deposit cash.
Use Cases	The next phase of the design process is to identify the use cases for the system being developed. • **Identify the Actors:** Identify the primary and secondary actors that will interact with the system. • **Identify the System Components:** The next step is to identify components of the system that will help in defining the use cases. • **Identify the Use Cases:** We identify the use cases that provide a good coverage to the requirements. • **Writing the Use Cases:** We now use textual modeling language FDL to define the use cases. The section also describes modeling of use cases with sequence diagrams. The following topics are covered: o Creating a Scenario Project o Representing the Actors and System Components o Entering the Use Case Interactions o Understanding Scenarios o Adding an Alternative Scenario to the Use Case o Adding More Scenarios o Choose the Level of Detail in Sequence Diagrams o Complete Example
Detailed Design	• **Object Sequence Diagrams:** The use case diagrams are transformed into object sequence diagrams that define the interactions in terms of public methods. • **Class Roles and Responsibilities:** The next step is to analyze and extract class design information from the object sequence diagrams. The following EventStudio diagrams are used in this step: o Interface Sequence Diagram o Object Wise Summary o Interface Collaboration Diagram • **Class Diagrams:** Armed with class level design information, we generate the class diagrams using Visual Studio 2005's built in class designer. • **Skeleton Source Code in C#:** Examine the skeleton code in C# to complete the journey from use cases to code.

© Copyright EventHelix.com Inc. All rights reserved.

2 Requirements

Here are the requirements for building an automated teller machine:

1.	The **ATM** ***customer*** interface shall be equipped with a: a. **Magnetic strip card reader** for swiping ATM cards. b. **Keypad** for user data entry. c. **Display** for user feedback. d. **Envelope dispensing slot** e. **Printer** for printing customer receipts
2.	The ATM keypad shall support the following keys: a. Numbers 0 to 9. b. "Enter" key to accept the transactions. c. "Cancel" key to cancel a transaction.
3.	The ATM shall communicate to the ***Bank*** via the Internet.
4.	The ATM **shall** perform the following authentication steps at the start of a **session**: a. The customer shall insert the ATM card. b. The ATM shall check the card integrity. c. The customer shall be prompted to enter the personal identification number (PIN). d. The ATM card number and the PIN shall be authenticated with the bank. e. The ATM shall abort the session if PIN authentication fails three times. (The PIN entered should match the PIN specified by the user.)
5.	The ATM shall perform the following end of session steps: a. The ATM card shall not be released at session end if PIN authentication fails. b. When a transaction ends in error, appropriate error message shall be displayed on the screen and the machine shall transition to the initial menu. c. The ATM shall issue a printed receipt to the customer at the end of a successful session. The receipt shall contain: • Date and time, • Machine location, • Account information, • Amount of the transaction • Available balance
6.	The ATM shall support the following **transactions**: a. Cash withdrawal from checking account. b. Cash deposit into customer checking account.
7.	The ATM shall perform the following transaction verification steps: a. Cash withdrawal shall be allowed only after the bank permits the transaction. b. Presence of a cash/check envelope shall be checked before completing the bank transaction. c. Cash and check deposits shall be manually verified to ascertain that the amount entered by the user matches the cash/check value in the deposit envelope. d. The money transfer between checking and saving shall be verified by the bank.
8.	The ATM shall support the following diagnostic features: a. **Logging** of all customer interactions b. Logging of all interactions with the bank c. The PIN for an ATM card shall not be logged.

© Copyright EventHelix.com Inc. All rights reserved.

3 Use Cases

The first step in the analysis is to develop a better understanding of the requirements by defining the use case diagrams.

3.1 Identify the Actors

Actors are entities that interact with the system under design.

The ATM is the system under design so we scan the requirements to identify entities that interact with the ATM. The actors have been marked in the requirements with ***bold-italics-underline***.

Customer	Customer needs to perform banking transactions using the ATM machine.
Bank	The bank serving this ATM.

3.2 Identify the System Components

The requirements also give us a view into the components and entities in the system under design. We have identified the system components in the requirements with **bold-underline**.

ATM	The ATM itself.
Card Reader	Card reader for swiping the ATM card.
Keypad	Keypad for entering numbers.
Display	Video display for the ATM
Envelope Dispenser	Envelope dispenser for accepting cash and checks.
Printer	Printer for printing the receipt for the user session.
Session	A session is initiated when the user swipes the ATM card and ends when the user indicates that he or she is done. Multiple transactions may be performed in a single session.
Transaction	A banking operation involving the bank is defined as a transaction. A transaction might involve cash withdrawal, cash/check deposit, money transfer and balance enquiry.
Logger	Logs the results of all sessions for auditing transactions.

3.3 Identify the Use Cases

Now we go through the requirements to identify the use cases that will give us a good coverage of the requirements:

- Customer withdraws cash
- Customer deposits cash
- Customer repeatedly enters invalid PIN

3.4 Writing the Use Cases

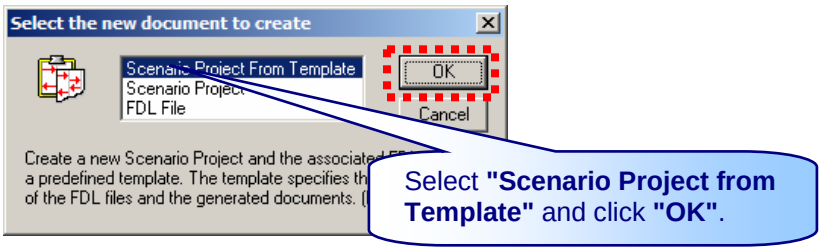
Let's start with the customer initiated use cases. We will be writing these use cases in FDL the simple text based modeling language used in EventStudio.

3.4.1 Creating a Scenario Project

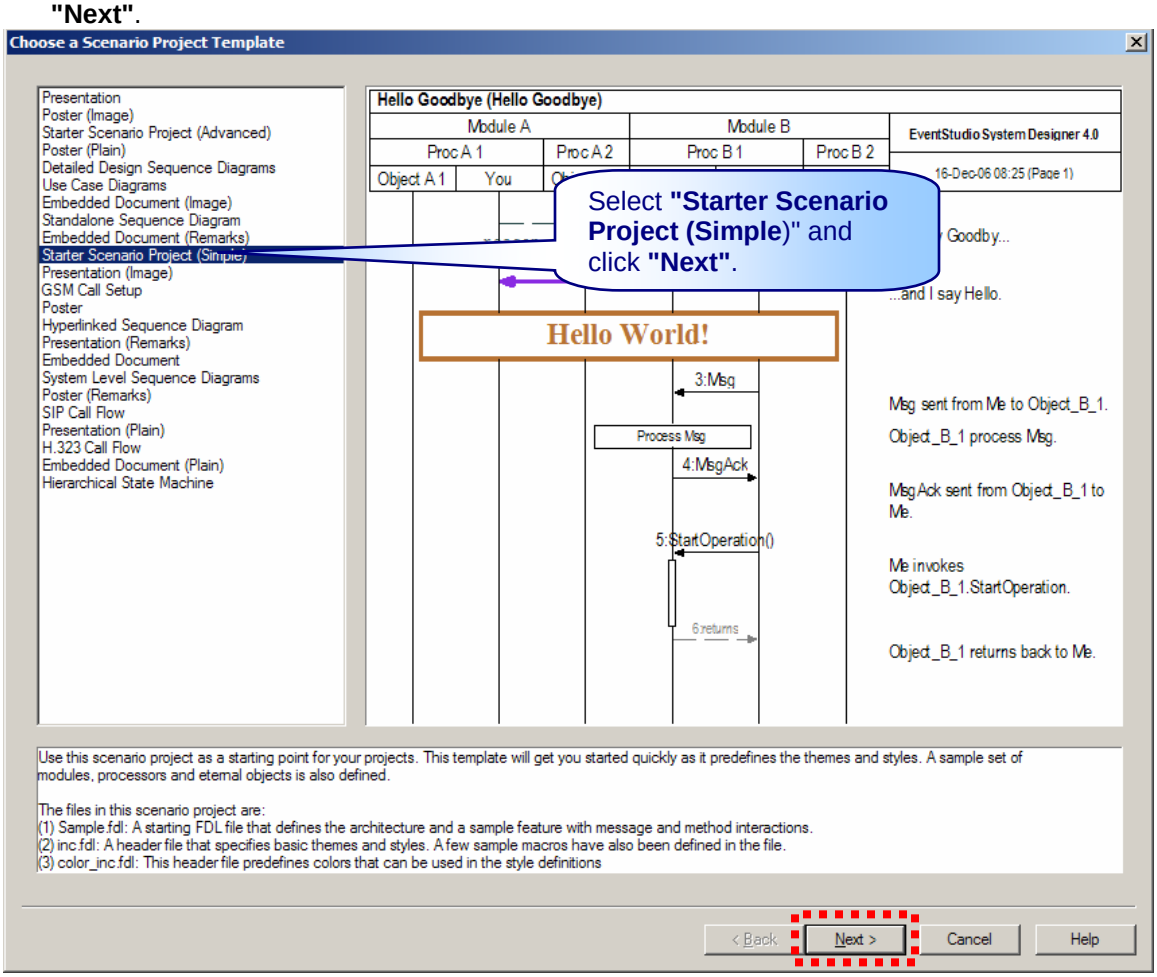
Before we get started with our use cases, we need to create a scenario project for the use cases. The following steps create a starter scenario project:

1. Invoke EventSudio from the Start Menu and select the **"Scenario Project from Template"** option.

© Copyright EventHelix.com Inc. All rights reserved.

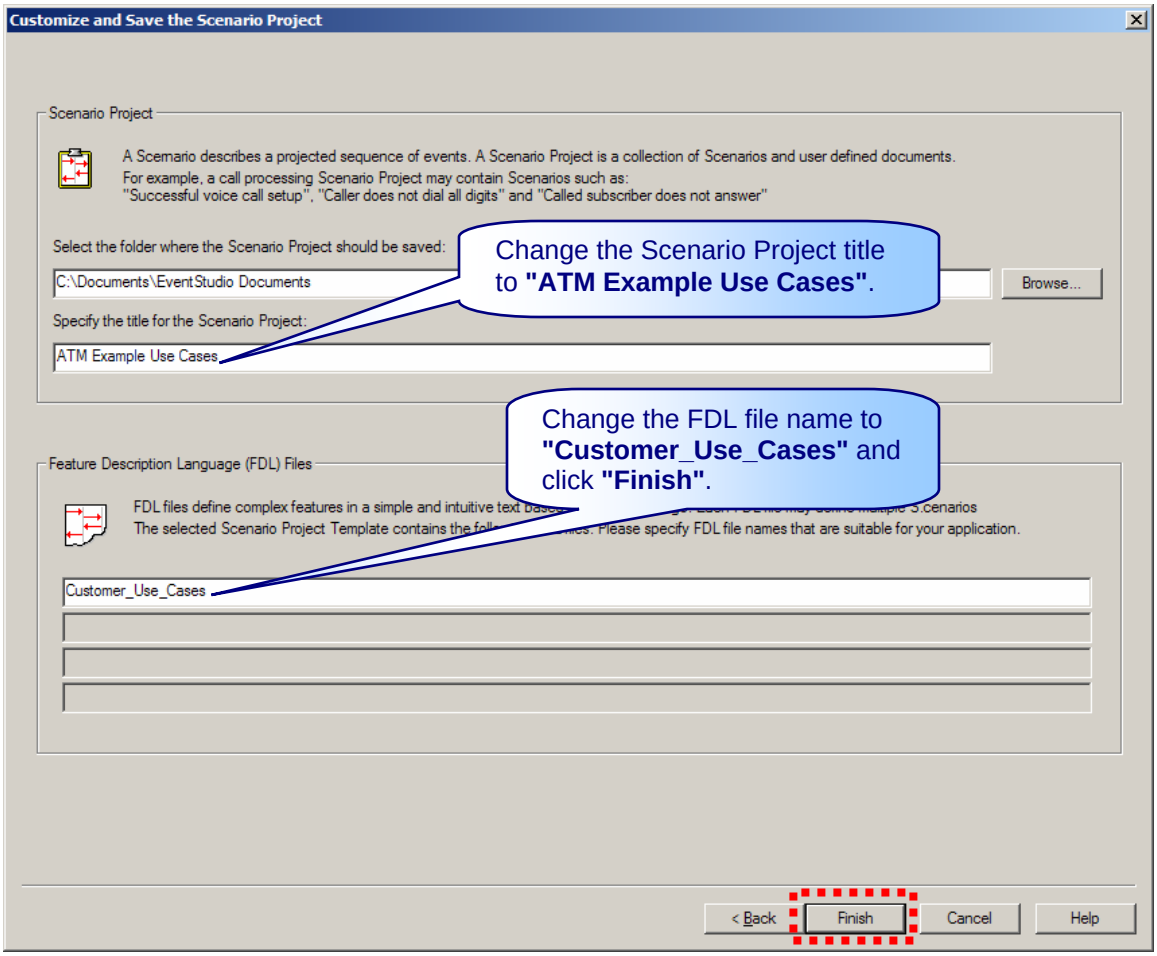


2. Select the **"Starter Scenario Project (Simple)"** from the displayed dialog. Then click **"Next"**.



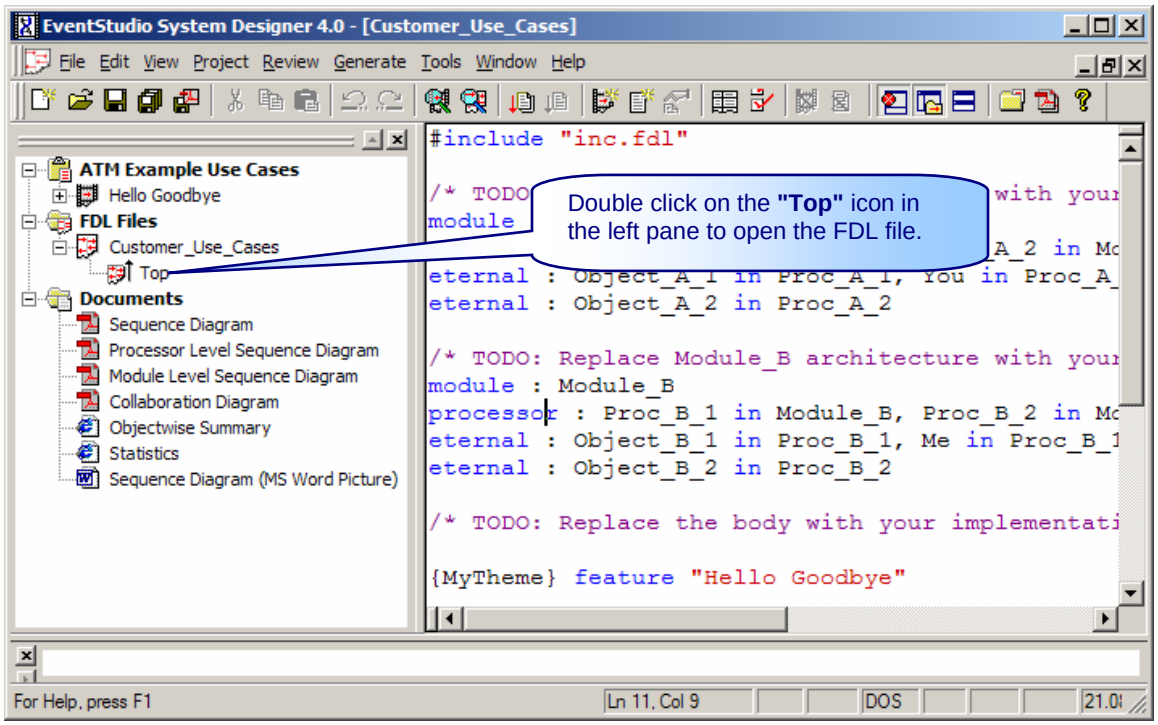
3. EventStudio will display another dialog.
   a. Change the title of the new scenario project of **"ATM Example Use Cases"**.
   b. Change the FDL file name to **"Customer_Use_Cases"**.

© Copyright EventHelix.com Inc. All rights reserved.



4. On clicking "Finish", EventStudio will generate and open a sequence diagram in Adobe Reader. Close this file.
5. Now move to the EventStudio main window and double click on the **"Top"** icon under the **"Customer_Use_Cases"** icon in the left pane (See the following figure). This opens **"Customer_Use_Cases.fdl"** file in the main workspace. We will be replacing the contents of this file in the following steps.

© Copyright EventHelix.com Inc. All rights reserved.



3.4.2 Representing the Actors and System Components

FDL allows you to partition your model into a three level hierarchy. At the highest level are modules. The system consists of modules. Modules contain processors and processors contain eternal and dynamic objects. We will be taking advantage of the hierarchical decomposition in the definition of the actors and the system.

We will decompose the use case entities into a three level hierarchy. At the highest level we have Actors and the System.

The second level further decomposes the system into smaller subdivisions. The Actors are further classified as Primary and Secondary Actors. The System is split into Input, Output and Controller.

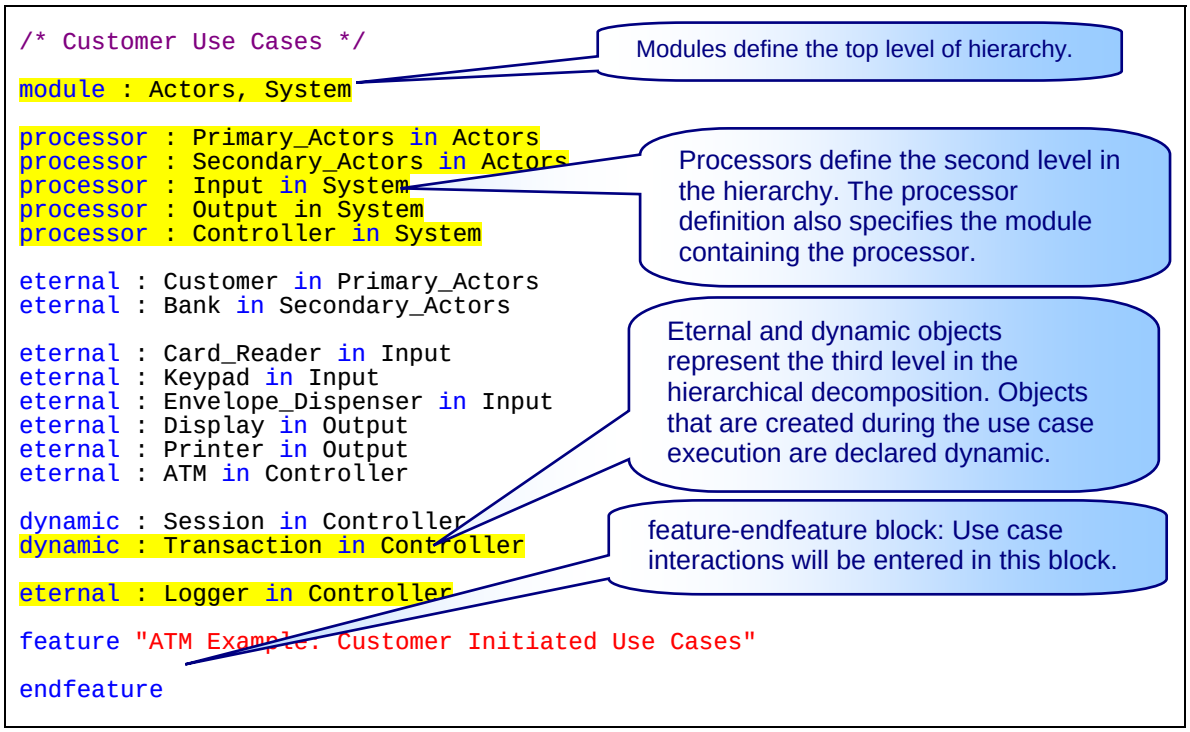
At the third level we will define eternal and dynamic objects. Eternal objects represent entities that exist throughout the use case life time. Entities that are instantiated during the use case are modeled as dynamic objects. Actors identified in section 3.1 are modeled as eternal objects in the Primary_Actors and Secondary_Actors processors. System components identified in section 3.2 are modeled as eternal and dynamic objects. Note that Session and Transaction are defined as dynamic objects as they will be created during the use case.

The final use case entity organization is shown below:

Actors		System								
Primary Actors	Secondary Actors	Input			Output		Controller			
Customer	Bank	Card Reader	Keypad	Envelope Dispenser	Display	Printer	ATM	Session	Transaction	Logger

Now let's map this into FDL:

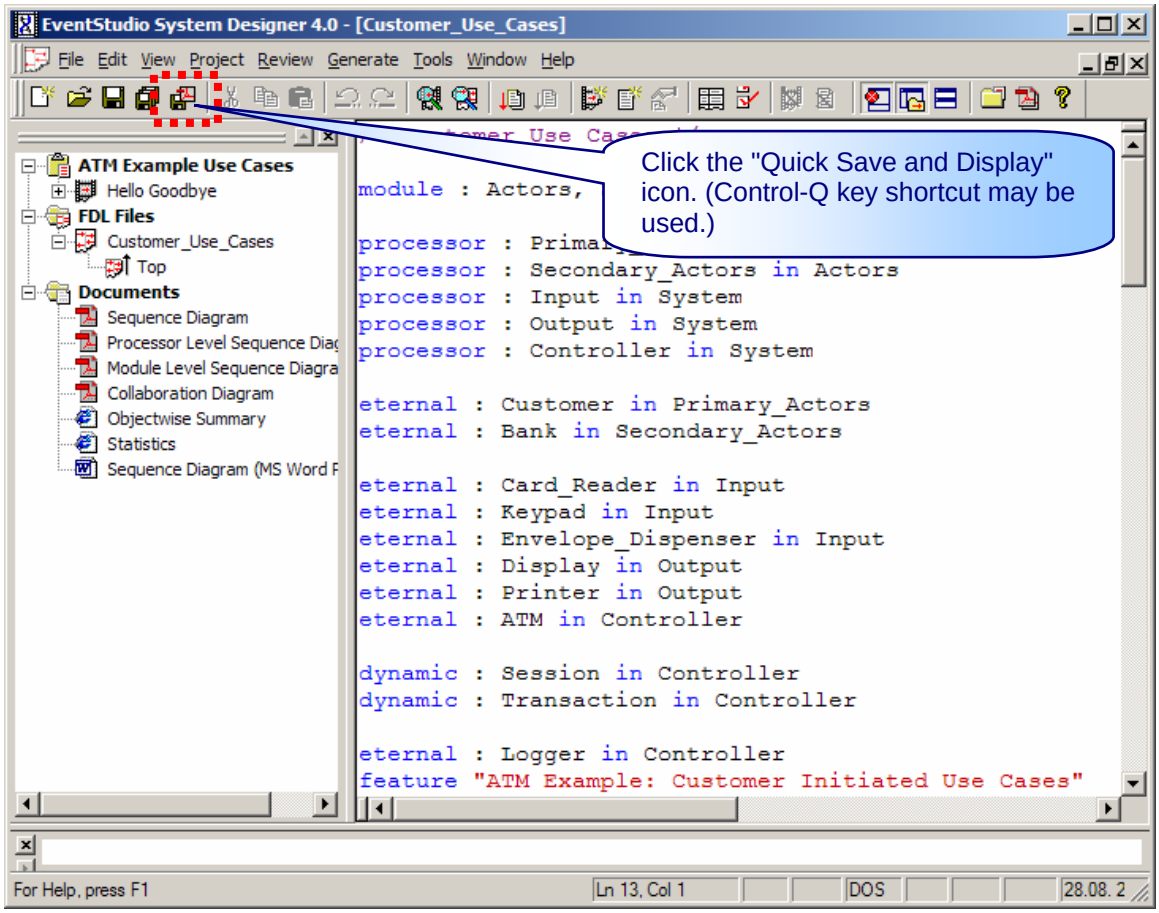
© Copyright EventHelix.com Inc. All rights reserved.



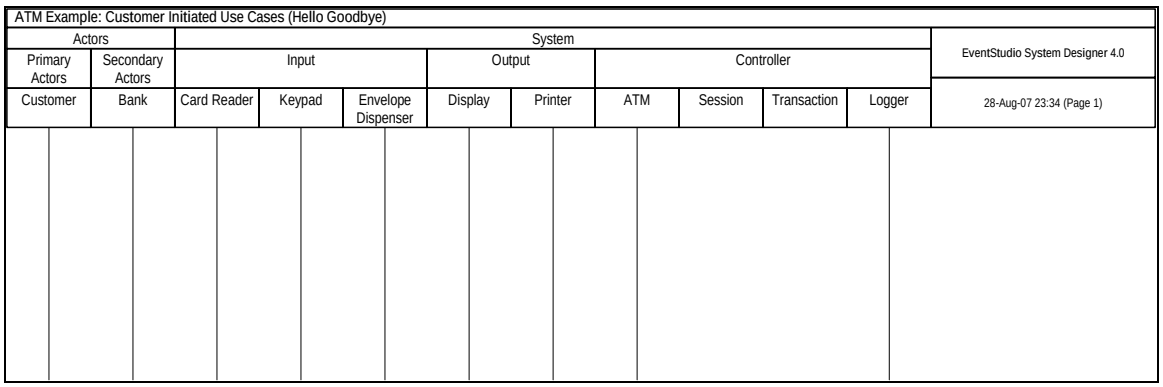
Now replace the contents of Customer_Use_Cases.fdl (see section 3.4.1) with the FDL shown above.

The next step is to click the **"Quick Save and Display"** icon (Control-Q is the keyboard shortcut).

© Copyright EventHelix.com Inc. All rights reserved.



EventStudio parses the FDL and generates a sequence diagram similar to the one shown below. Note that this diagram defines the entities organized hierarchically. An axis is drawn for each eternal object.



We have defined the architecture of the use case components. The next section describes modeling of the interactions in the use case diagram.

3.4.3 Entering the Use Case Interactions

Now lets define the core use case that handlers a customer session and transactions. This use case will define the basis for the customer initiated use cases identified in section 3.3.

© Copyright EventHelix.com Inc. All rights reserved.

We scan the requirements in section 1 to specify the interactions involved in setting up an ATM session followed by a single generic transaction.

The FDL representation of the use case is shown below. We have highlighted some of the important statements.

```
feature "ATM Example: Customer Initiated Use Cases"

  heading "Session start handling"

  "Insert Credit Card" : Customer -> Card_Reader
  (* The customer inserts the ATM card to start the ATM session. *)

  "Credit Card Swipe" ("ATM Card Number", PIN, ...) : Card_Reader -> ATM
  (* The Card Reader in the ATM detects the credit
the ATM object. *)

  ATM creates Session
  (* The ATM object has detected the start of a new sessi
creates a new Session object. *)

  Initialize ("ATM Card Number", PIN): ATM -> Session
  (* Update the Session object with the ATM card number and PIN stored on the
ATM card. *)

  chain
     "Welcome Message" : Display <- Session
     "Welcome Message" : Customer <- Display
  endchain
  (* A welcome message is displayed to the user. *)

  chain
     "Request PIN" : Display <- Session
     "Request PIN Box Display" : Customer <- Display
  endchain
  (* Request the user to enter the PIN. *)

  chain
     "Enter PIN" : Customer -> Keypad
     "Handle PIN Entry" : Keypad -> Session
  endchain
  (* The Customer enters the PIN. *)

  Session takes action "Compare PIN read from the card with the entered PIN"
  (* Does the PIN entered by the user match the PIN stored in the card? *)

  chain
     "Display Transaction Menu" : Display <- Session
     "Display Transaction Menu" : Customer <- Display
  endchain
  (* The PIN has matched, so give the transaction choices to the user. *)

  chain
     "Select Transaction Menu" : Customer -> Keypad
     "Handle Transaction Selection" : Keypad -> Session
  endchain
  (* User selects the transaction. *)

  heading "General handling for a transaction"
  Session creates Transaction
  (* The Session creates the Transaction object. *)

  Transaction takes action "Perform Transaction"
  (* Transaction is processed. *)

  "Transaction Log" : Transaction -> Logger
  "Transaction Complete" : Session <- Transaction
  (* Notify the session object that the transaction has been completed. *)
```

A heading definition.

An interaction from the "Customer" actor to the Card_Reader object. A remark is enclosed within (* and *).

A message interaction chain with two messages.

Session object is creating a dynamic Transaction object.

© Copyright EventHelix.com Inc. All rights reserved.

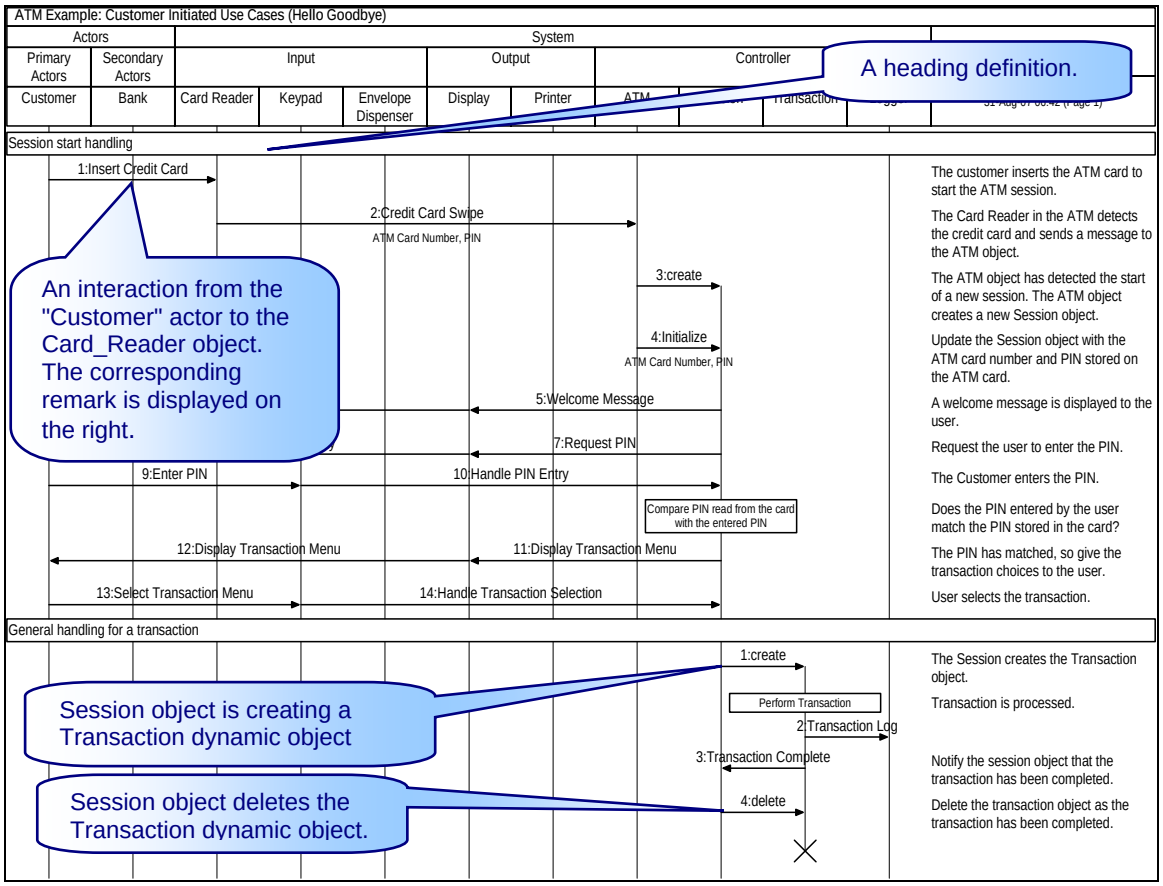
```
  Session deletes Transaction
  (* Delete the transaction object as the transaction has been completed. *)

  chain
     "Display Transaction Menu" : Display <- Session
     "Display Transaction Menu" : Customer <- Display
  endchain
  (* Again display the transaction menu, giving the user the option to initiate
another transaction. *)

  chain
     "Select Transaction Menu" : Customer -> Keypad
     "Handle Transaction Selection" : Keypad -> Session
  endchain
  (* The user makes a selection. In this case, the user decides to end the
session. *)

  heading "Session end handling"

  Session takes action "Ending Session"

  "Eject ATM Card" : Card_Reader <- Session
  (* Eject the ATM card as the session has been completed successfully. *)

  "Session Log" : Session -> Logger
  (* Update the session level log. *)

  "Session Complete" : ATM <- Session
  (* Notify the ATM object. *)

  ATM deletes Session
  (* ATM deletes the Session object. *)

endfeature
```

Session object deletes the Transaction dynamic object.

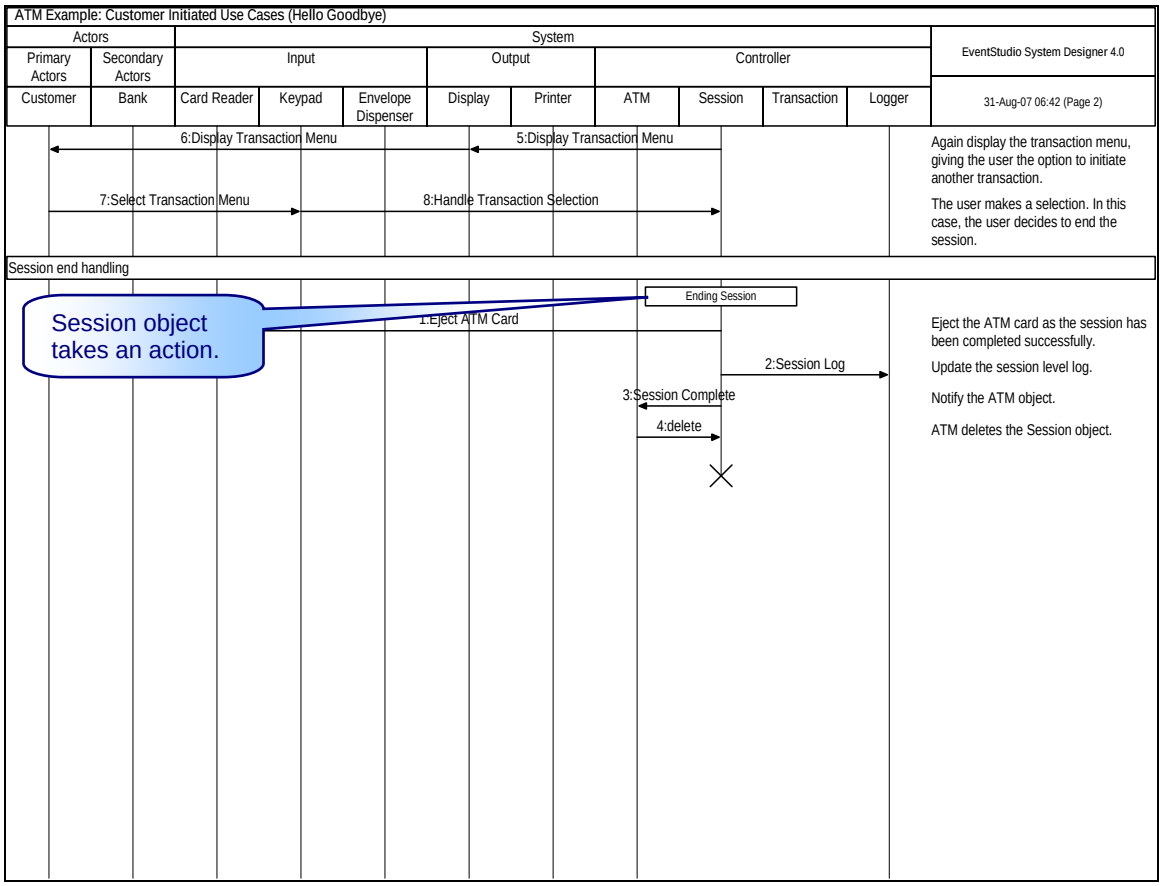
Session object takes an action.

The next step is to click the **"Quick Save and Display"** icon (Keyboard: Control-Q). EventStudio will generate a PDF sequence diagram similar to the one shown below. You can see the representation of the highlighted FDL statements in the sequence diagram.

© Copyright EventHelix.com Inc. All rights reserved.



© Copyright EventHelix.com Inc. All rights reserved.



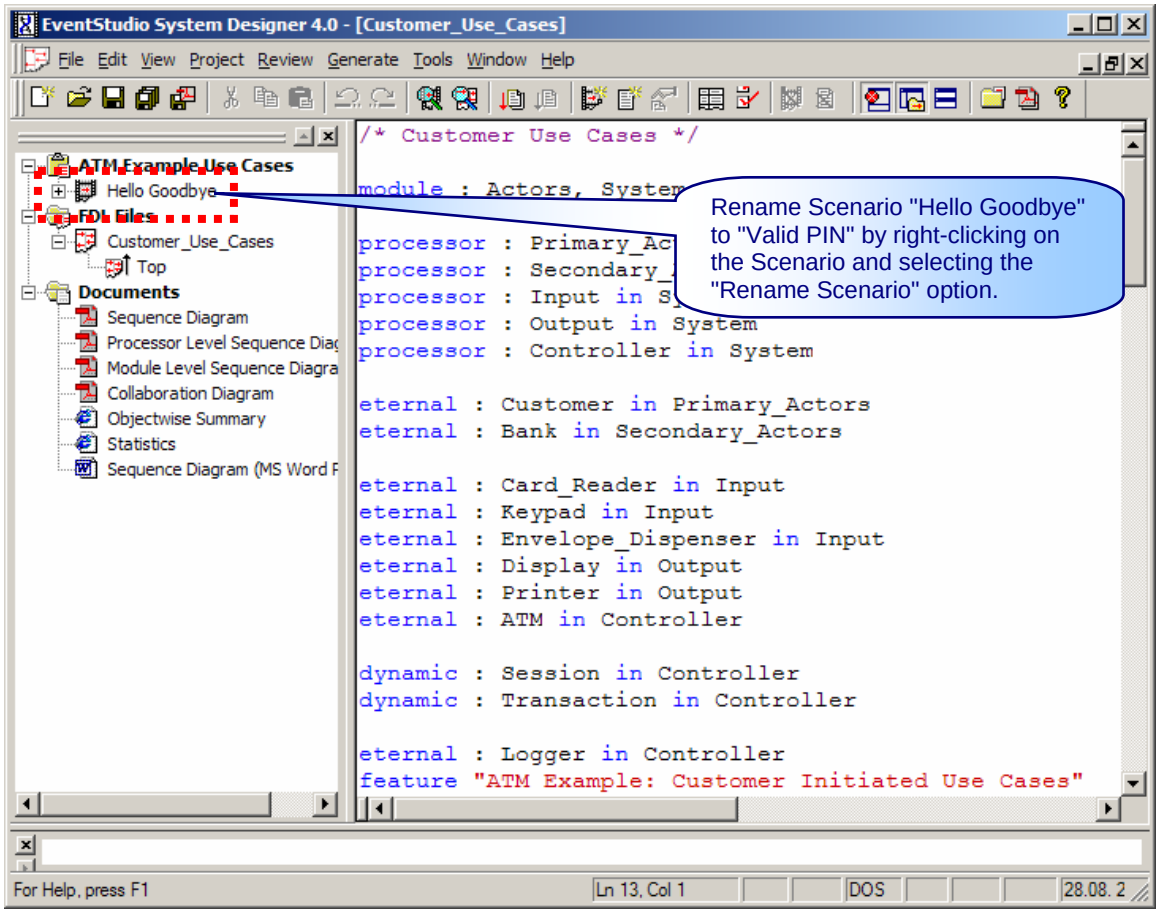
3.4.4 Understanding Scenarios

Before we move further, let's look at Scenarios in EventStudio. A single FDL file supports definition of multiple scenarios. The differences in the scenarios are identified using the `case-leg-endcase` and `if-else-endif` statements (covered later). We have not used these statements as we have only covered one scenario.

The scenario is represented in the left side scenario bar as shown below. EventStudio template we used created a default scenario called "Hello Goodbye". Let's rename this scenario to "Valid PIN" by right-clicking on the Scenario and selecting the "Rename Scenario" command.

This scenario will serve as our successful case scenario. Addition of `case-leg-endcase` statements will result in modification to this scenario to select the `case-leg`s that apply to this scenario.

© Copyright EventHelix.com Inc. All rights reserved.



3.4.5 Adding an Alternative Scenario to the Use Case

We have defined a use case for a successful ATM session. We will now add an alterative scenario. The new scenario will cover the handling of an ATM session when the user enters an invalid PIN.

The new scenario is added by simply introducing a `case-endcase` block with `leg` statements for valid and invalid PIN scenarios.

```
feature "ATM Example: Customer Initiated Use Cases"

  heading "Session start handling"

  "Insert Credit Card" : Customer -> Card_Reader
  (* The customer inserts the ATM card to start the ATM session. *)

  "Credit Card Swipe" ("ATM Card Number", PIN) : Card_Reader -> ATM
  (* The Card Reader in the ATM detects the credit card and sends a message to
the ATM object. *)

  ATM creates Session
  (* The ATM object has detected the start of a new session. The ATM object
creates a new Session object. *)

  Initialize ("ATM Card Number", PIN): ATM -> Session
  (* Update the Session object with the ATM card number and PIN stored on the
ATM card. *)

  chain
     "Welcome Message" : Display <- Session
```


© Copyright EventHelix.com Inc. All rights reserved.

```
    "Welcome Message" : Customer <- Display
  endchain
  (* A welcome message is displayed to the user.

  chain
     "Request PIN" : Display <- Session
     "Request PIN Box Display" : Customer <- Disp
  endchain
  (* Request the user to enter the PIN. *)

  chain
     "Enter PIN" : Customer -> Keypad
     "Handle PIN Entry" : Keypad -> Sess
  endchain
  (* The Customer enters the PIN. *)

  Session takes action "Compare PIN read from the C    with the entered PIN"
  (* Does the PIN entered by the user match th    IN stored in the card? *)

  case
     leg "Valid PIN":

     leg "Invalid PIN":
        [* Begin Loop: Retry PIN *]
        chain
           "Request PIN Reentry" : Display <- Session
           "Invalid PIN. Please Enter PIN Again" : Customer <- Display
        endchain

        chain
            "Enter PIN" : Customer -> Keypad
            "Handle PIN Entry" : Keypad -> Session
        endchain
        [* End Loop: Retry PIN *]

        "Swallow ATM Card": Card_Reader <- Session
        (* PIN entry has failed, swallow the ATM card *)

        Card_Reader takes action "Swallow Card"

        goto exit
  endcase

  chain
     "Display Transaction Menu" : Display <- Session
     "Display Transaction Menu" : Customer <- Display
  endchain
  (* The PIN has matched, so give the transaction choices to the user. *)

                              ...Other Statements...

  ATM deletes Session
  (* ATM deletes the Session object. *)

label exit:

endfeature
```

The case-endcase block defines two legs. The "Valid PIN" leg adds no specific actions and continues execution after endcase.

The "Invalid PIN" retries PIN entry and swallows the ATM card on repeated failures.

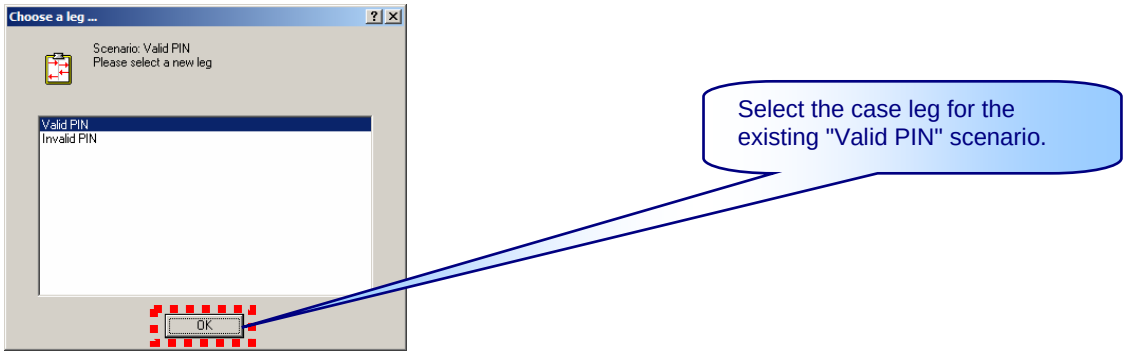
A block remark

Skip the session handling logic and jump straight to the exit label.

The exit label

Now let's save and regenerate the use case diagram by **Control-Q** (Quick Save and Display). EventStudio senses the new case statement and asks us to make the leg selection for the existing Scenario ("Valid PIN"). We select "Valid PIN" leg for the "Valid PIN" Scenario.

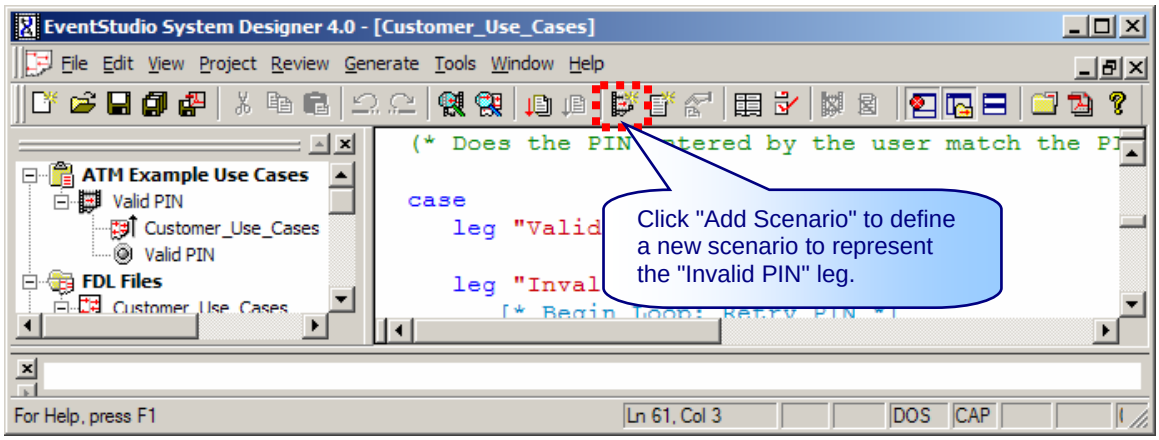
© Copyright EventHelix.com Inc. All rights reserved.



Once the selection is made, EventStudio associates the case leg "Valid PIN" with the "Valid PIN" scenario.

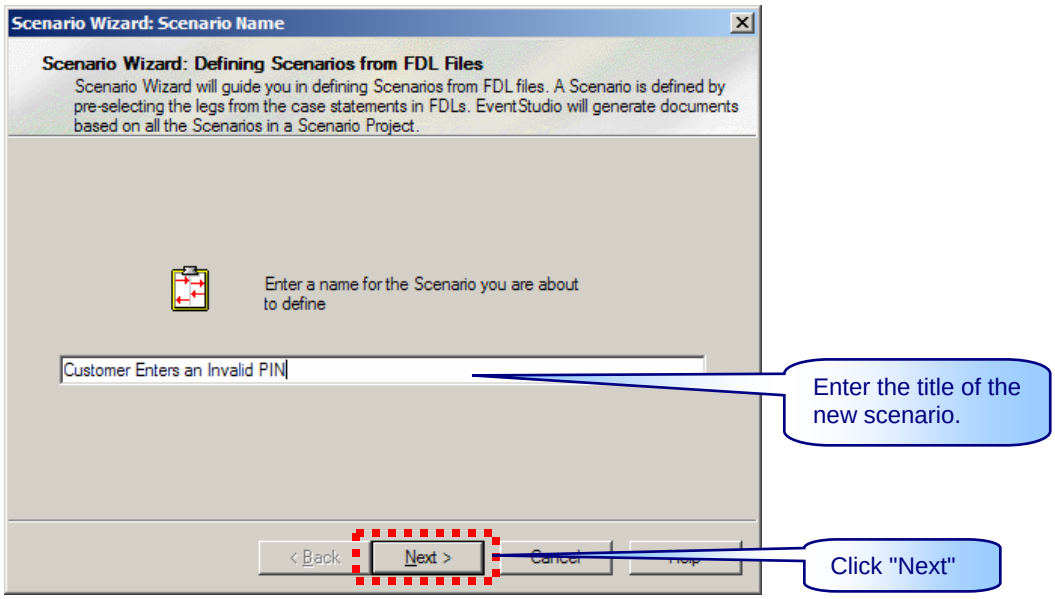
We will now add a second scenario for the case where the user enters an invalid PIN. The steps involved here are:

1. Click on the **"Add Scenario" (Control-R).**

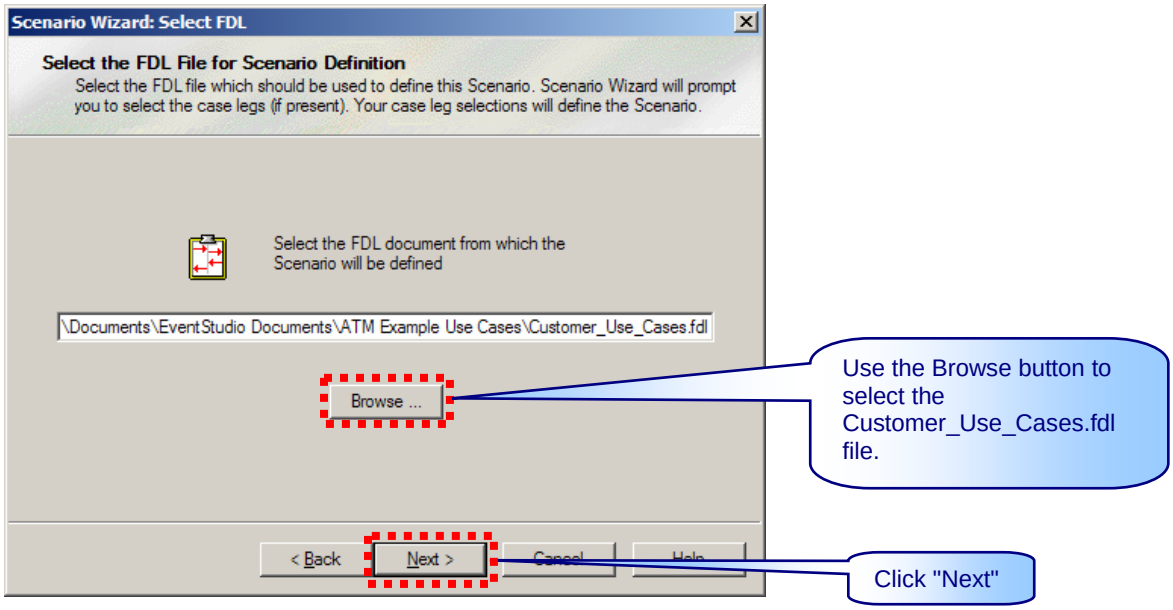


2. Enter the Scenario title **"Customer Enters an Invalid PIN"** and click **"Next".**

© Copyright EventHelix.com Inc. All rights reserved.

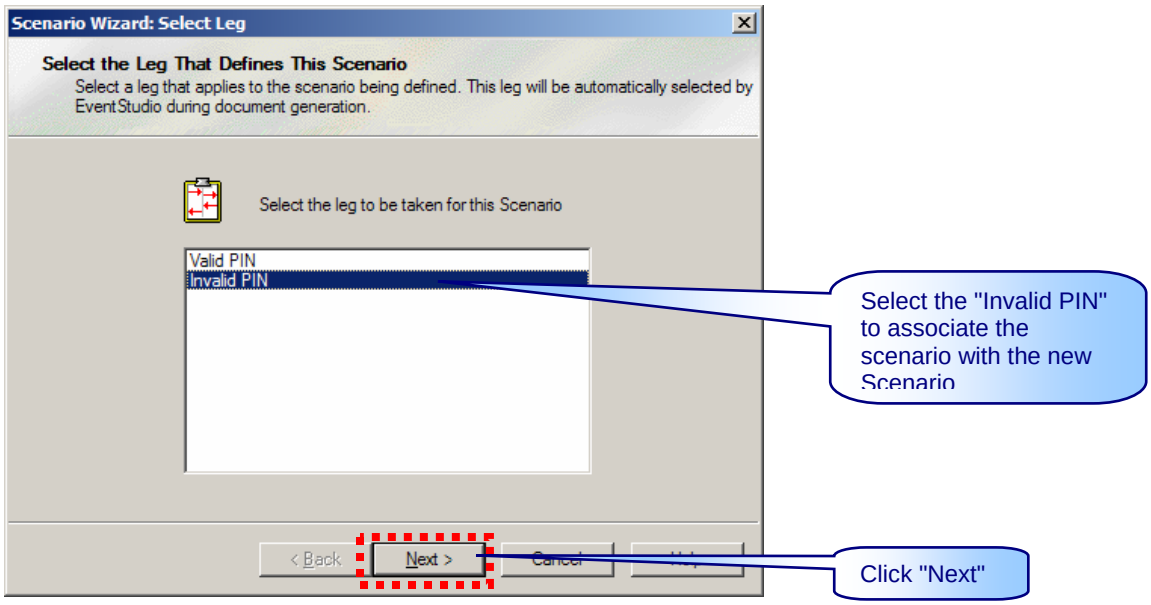


3. Select the **"Customer_Use_Cases.fdl"** file and click **"Next".**

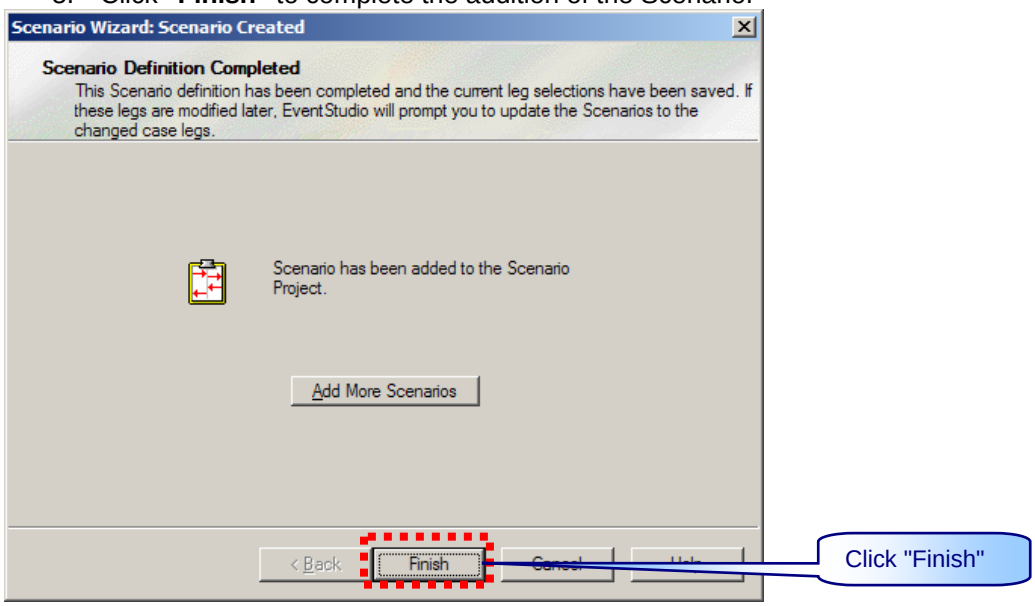


4. Now select the **"Invalid PIN"** leg and click **"Next"**.

© Copyright EventHelix.com Inc. All rights reserved.

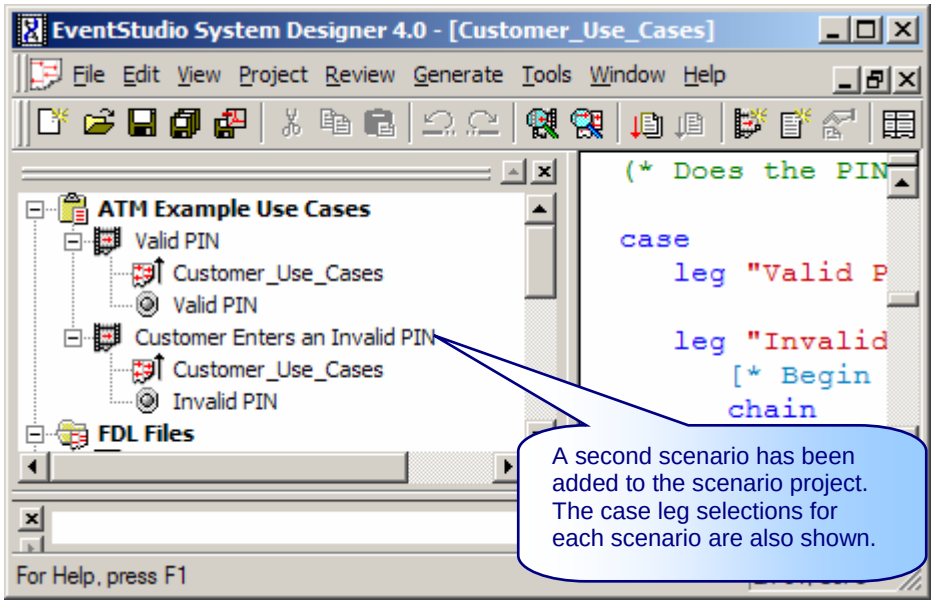


5. Click **"Finish"** to complete the addition of the Scenario.

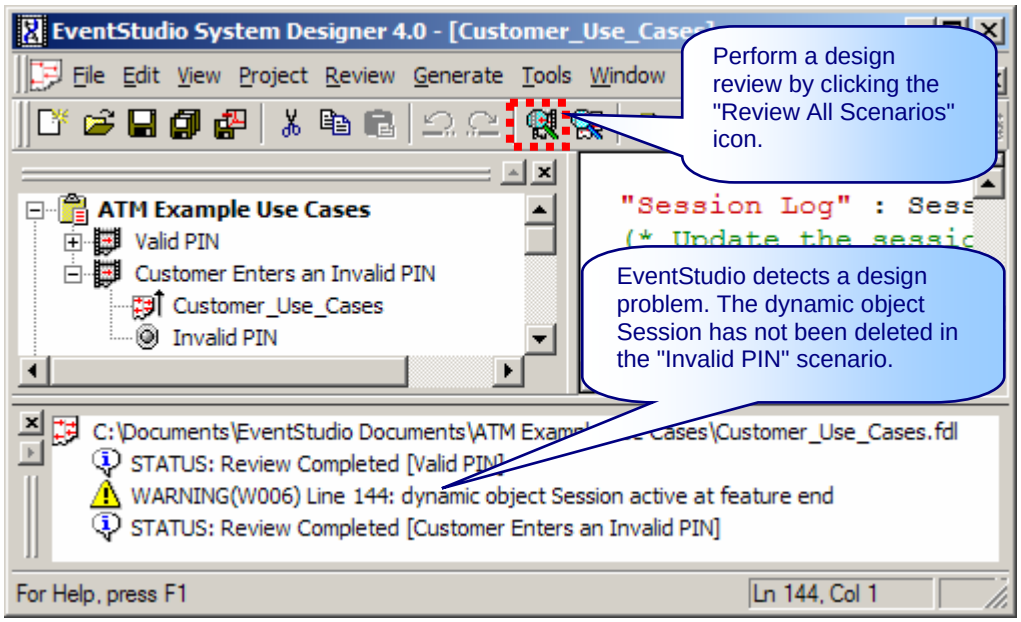


6. EventStudio now displays two Scenarios in the left pane. The legs uniquely identifying the Scenarios are also listed.

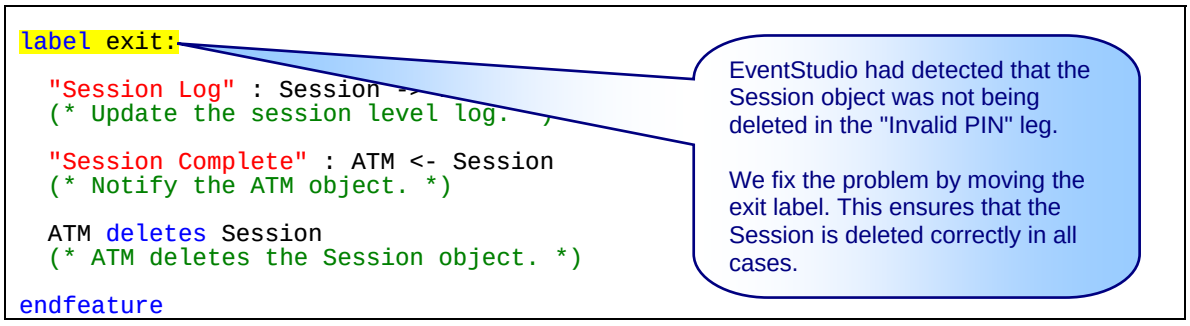
© Copyright EventHelix.com Inc. All rights reserved.



7. Now click the "Review All Scenarios" icon (**Control-W**) to look for design errors.

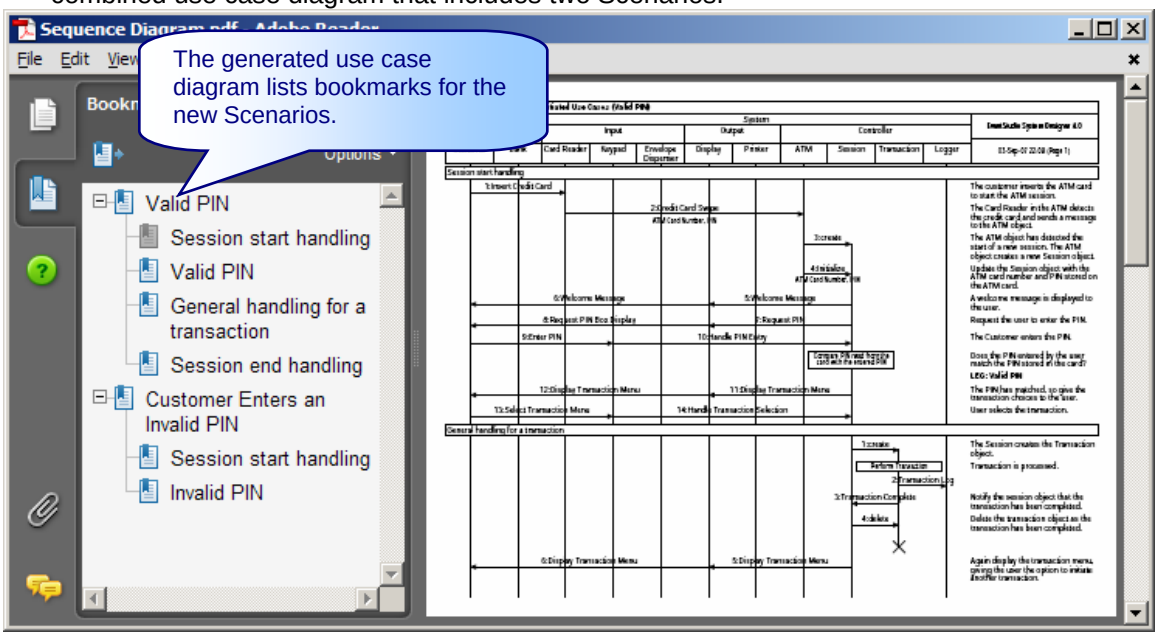


8. We fix the problem by moving the exit label.



© Copyright EventHelix.com Inc. All rights reserved.

9. Now click **Control-Q** to save and generate the documents. EventStudio generates a combined use case diagram that includes two Scenarios.



3.4.6 Adding More Scenarios

We now expand the use case by adding the implementation for the "cash withdrawal" and "cash deposit" scenarios.

The scenario support is added in the FDL is via a case-endcase statement. An important point to note here is the use of #includes and #define-macros to simplify the FDL.

Once the scenarios have been added type **"Control+Q"** to update the sequence diagram.

```
/* == inc.fdl == */

/* -- Colors -- */
#include "colors_inc.fdl"

/* -- Useful Macros -- */

#define TRANSACTION_INPUT(_command, _via)   chain;\
                                    _command : Customer -> _via;\
                                    _command : _via -> Transaction;\
                                endchain

#define TRANSACTION_OUTPUT(_command, _via)   chain;\
                                    _command : _via <- Transaction;\
                                    _command : Customer <- _via;\
                                endchain
```

C preprocessor type macros are defined to handle repetitive patterns. Multi-line macros can be defined by separating lines with **;**.

```
#include "inc.fdl"

/* Customer Use Cases */

module : Actors, System

                    ...Other Declarations...
```

Include the "inc.fdl" defined above.

© Copyright EventHelix.com Inc. All rights reserved.

```
feature "ATM Example: Customer Initiated Use Cases"

                                ...Other Statements...

  Session creates Transaction
  (* The Session creates the Transaction object. *)

  case
     leg "Cash Withdrawal from Checking Account":

          TRANSACTION_OUTPUT("Enter the Amount to be Withdrawn", Display)
          (* Request the customer to enter the amount to be withdrawn. *)

          TRANSACTION_INPUT("Dollar Amount", Keypad)
          (* The customer keys in the dollar amount that he or she wishes to withdraw. *)

          "Authorize Cash Withdrawal"("ATM card #", PIN, Amount): Bank <- Transaction
          (* Request the Bank to authorize the cash withdrawal. *)

          Bank takes action "Validate the ATM card, PIN and check Amount availability"
          (* The Bank validates the transaction. *)

          "Withdrawal Accepted" (Amount): Bank -> Transaction
          (* Bank allows the cash withdrawal. *)

          TRANSACTION_OUTPUT("Cash", Cash_Dispenser)
          (* Dispense the cash to the user. *)

     leg "Cash Deposit into Checking Account":

          TRANSACTION_OUTPUT("Enter Cash Amount for deposit", Display)
          (* Ask the user to enter the cash amount for deposit. *)

          TRANSACTION_INPUT("Cash Amount", Keypad)
          (* The customer enters the cash amount. *)

          TRANSACTION_OUTPUT("Request Cash in Envelope", Display)
          (* Request the user to prepare an envelope with cash. *)

          Customer takes action "Keep cash in envelope"

          "Envelope With Cash" : Customer -> Envelope_Slot
          (* Customer inserts the envelope. *)

          "Received Envelope" : Envelope_Slot -> Transaction
          (* Envelope Slot informs the Transaction object that the Envelope has been
          successfully received. *)

          TRANSACTION_OUTPUT("Deposit Success", Display)
          (* Inform the user that the deposit envelope has been accepted by the ATM. *)

  endcase

                                ...Other Statements...

endfeature
```

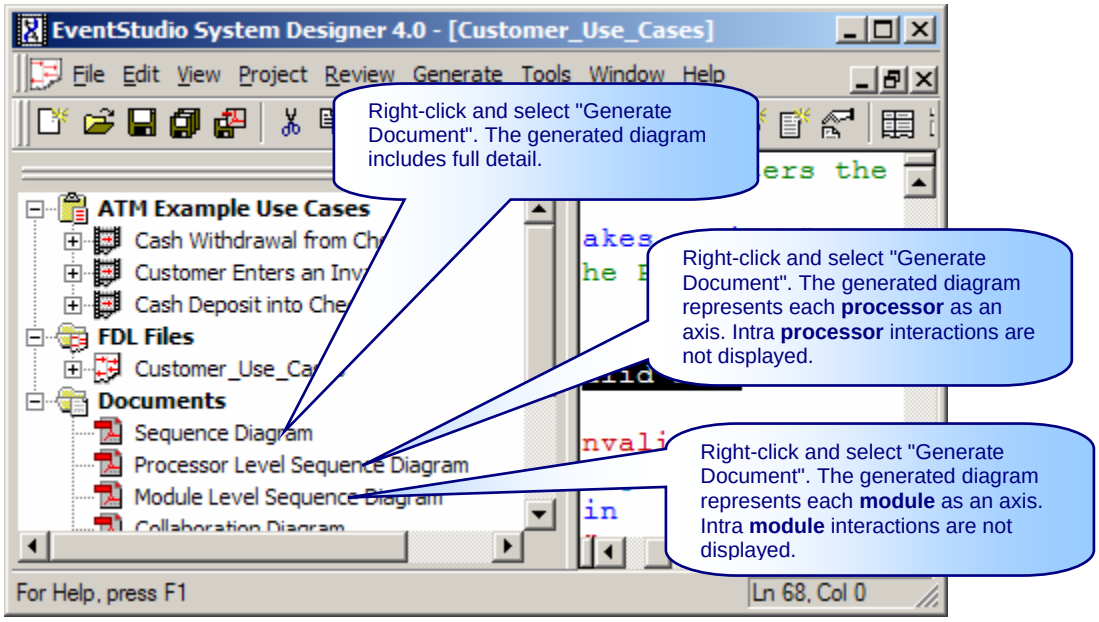
Using the macros that we defined in **"inc.fdl"** include file.

3.4.7 Choose the Level of Detail in Sequence Diagrams

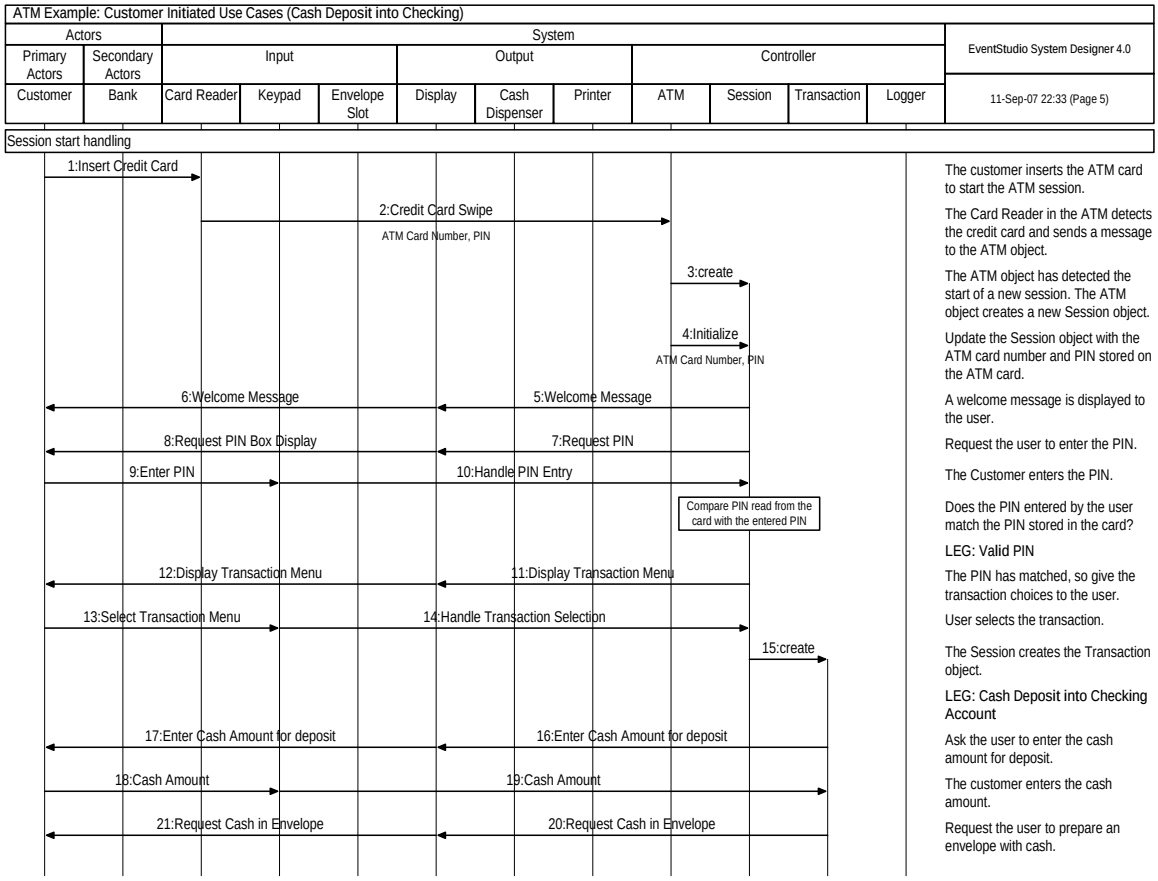
EventStudio lets you control the level of detail in the generated sequence diagrams. The different levels of detail are listed below.

Diagram type	Level of Detail	How to Generate	Example
Sequence Diagram	Sequence diagram with full detail.	Right Click on the "Sequence Diagram" in the Left Pane and select "Generate Document".	3.4.7.1
Processor Interaction Sequence Diagram	Intra processor details are hidden. Only inter processor interactions are shown.	Right Click on the "Processor Level Sequence Diagram" in the Left Pane and select "Generate Document".	3.4.7.2
Module Interaction Sequence Diagram	Intra module details are hidden. Only inter module interactions are shown.	Right Click on the "Module Level Sequence Diagram" in the Left Pane and select "Generate Document".	3.4.7.3

© Copyright EventHelix.com Inc. All rights reserved.

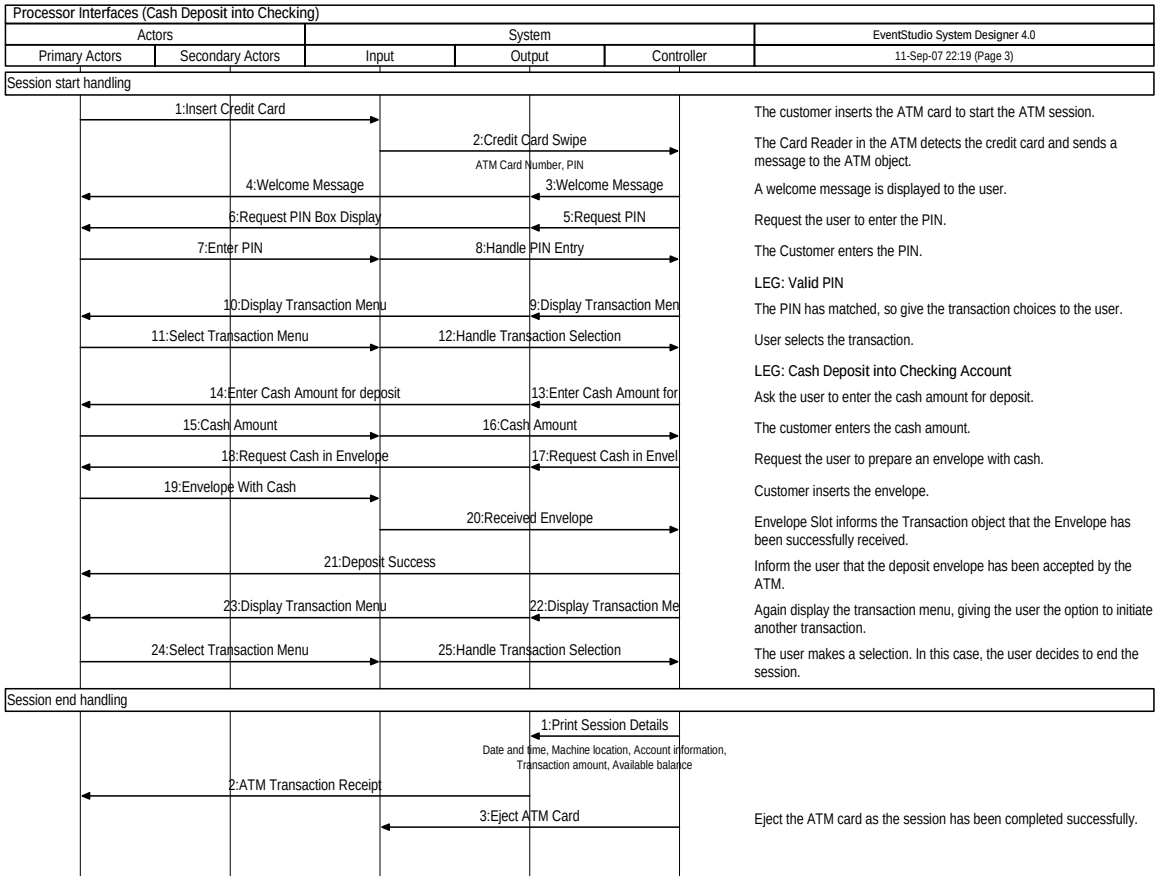


3.4.7.1 Sequence Diagram

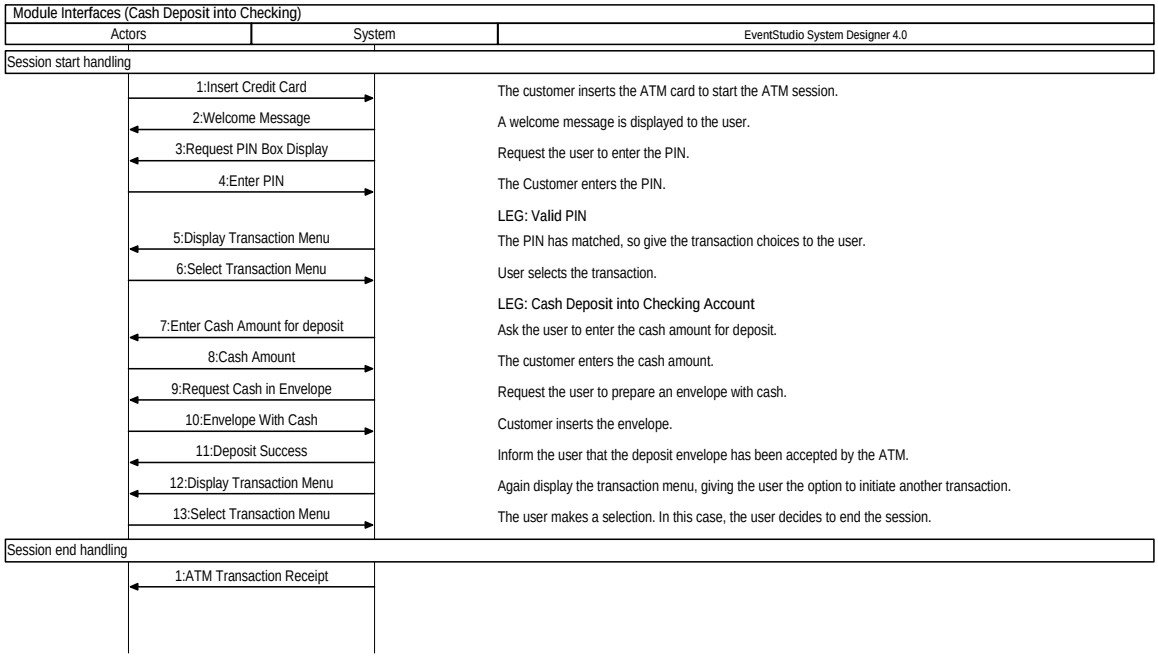


© Copyright EventHelix.com Inc. All rights reserved.

3.4.7.2 Processor Interaction Sequence Diagram



3.4.7.3 Module Interaction Sequence Diagram



© Copyright EventHelix.com Inc. All rights reserved.

3.4.8 Complete Example

The complete example can be downloaded from:
http://www.EventHelix.com/EventStudio/Tutorial/Use_Case_Sequence_Diagram_Tutorial.zip

© Copyright EventHelix.com Inc. All rights reserved.

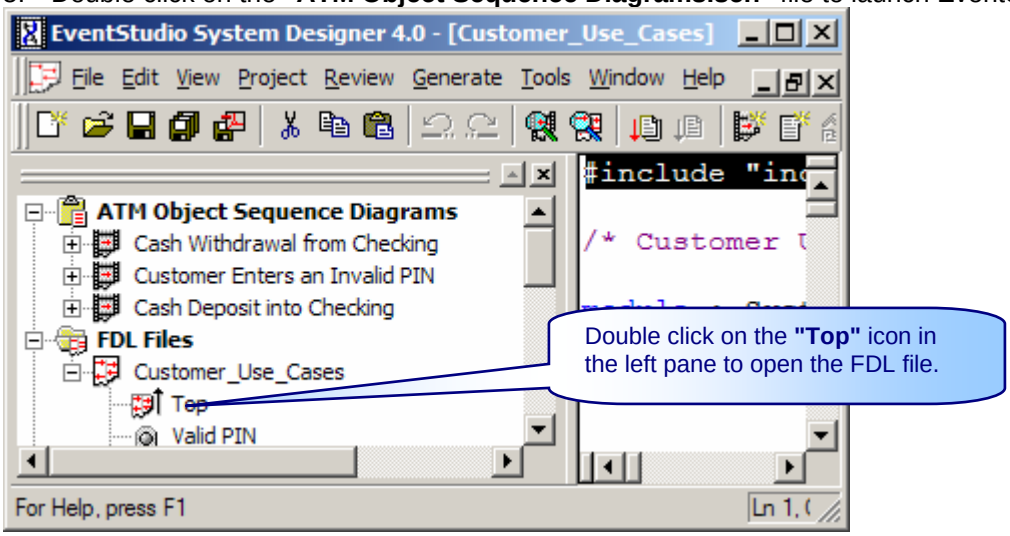
4 Detailed Design

The first phase of analysis and design has been completed with the generation of detailed use case diagrams. We will now focus on the detailed design.

4.1 Object Sequence Diagrams

The first step in class design is to convert the use case diagrams into object sequence diagrams. The sequence diagrams will add method level detail. The design steps are:

1. Navigate to the **"EventStudio Documents"** folder in your **"My Documents"** folder.
2. Make a copy of the **"ATM Example Use Cases"**.
3. Rename the copied folder as **"ATM Object Sequence Diagrams"** and navigate to the folder.
4. Inside the folder rename the scenario project **"ATM Example Use Cases.scn"** to **"ATM Object Sequence Diagrams.scn"**.
5. Double click on the **"ATM Object Sequence Diagrams.scn"** file to launch EventStudio.



6. We go through the use case diagram and convert the message interactions into method invocation. The structure of the FDL is also changed. More details are added. The new FDL file is shown below[1]:

```
#include "inc.fdl"

module : System
processor : ATM_Controller in System
eternal : Bank in ATM_Controller
eternal : Card_Reader in ATM_Controller
eternal : Keypad in ATM_Controller
eternal : Envelope_Slot in ATM_Controller
eternal : Display in ATM_Controller
eternal : Cash_Dispenser in ATM_Controller
eternal : Printer in ATM_Controller
eternal : ATM in ATM_Controller
dynamic : Session in ATM_Controller
dynamic : Transaction in ATM_Controller
eternal : Logger in ATM_Controller

feature "ATM Example: Customer Initiated Use Cases"

  "Insert Credit Card" : env_l -> Card_Reader
  (* The customer inserts the ATM card to start the ATM session. *)

  Card_Reader invokes ATM.HandleCreditCardSwipe(cardNumber, pin)
  (* The Card Reader in the ATM detects the credit card and sends a message to the ATM
object. *)
```

The customer has been represented as an environment. FDL supports env_l and env_r as the left and right environment.

[1] The complete example can be downloaded from [TBD].

© Copyright EventHelix.com Inc. All rights reserved.

```
    ATM creates Session
    (* The ATM object has detected the start of a new session. The ATM object creates a
new Session object. *)

    ATM invokes Session.Start (cardNumber, pin)
    (* Update the Session object with the ATM card # and PIN stored on the ATM card. *)
      Session invokes Display.Print("Welcome to the XYZ Bank ATM")
      (* A welcome message is displayed to the user. *)
      Display.Print returns
      Session invokes Display.Print("Please enter the PIN")
      (* Request the user to enter the PIN. *)
      Display.Print returns
    Session.Start returns

  ATM.HandleCreditCardSwipe returns

  "Enter PIN" : env_l -> Keypad   (* The Customer enters the PIN. *)
  Keypad invokes Session.HandlePIN(pin)

    Session takes action "Compare PIN read from the card with the entered PIN"
    (* Does the PIN entered by the user match the PIN stored in the card? *)

    case
       leg "Valid PIN":

       leg "Invalid PIN":
          [* Begin Loop: Retry PIN expired *]
          Session invokes Display.Print("Collect your ATM card from the bank")
          Display.Print returns
          Session invokes Card_Reader.SwallowCard
          Card_Reader.SwallowCard returns
          Session invokes Session.End(InvalidATM)
             Session invokes ATM.HandleSessionComplete
                 ATM takes action "Mark Session for Deletion"
             ATM.HandleSessionComplete returns
          Session.End returns
          Session.HandlePIN returns
          Session deletes Session

          goto exit
    endcase
    Session invokes Display.Print("(1) Deposit (2) Withdraw (3) Quit")
    (* The PIN has matched, so give the transaction choices to the user. *)
    Display.Print returns
  Session.HandlePIN returns

  "Select Transaction Menu" : env_l -> Keypad
  (* User selects the transaction. *)
  Keypad invokes Session.HandleTransactionSelection(transaction_type)
   case
      leg "Cash Withdrawal from Checking Account":
        Session creates Transaction
        (* The Session creates the WithdrawCashTransaction object. *)

        Session invokes Transaction.Start
          Transaction invokes Display.Print("Enter the Amount to be Withdrawn")
          (* Request the customer to enter the amount to be withdrawn. *)
          Display.Print returns
        Transaction.Start returns
        Session.HandleTransactionSelection returns

        "Dollar Amount" : env_l -> Keypad
        Keypad invokes Transaction.HandleAmountEntry(amount)
        (* The customer keys in the dollar amount that he or she wishes to withdraw. *)
           Transaction invokes Bank.RequestWithdrawalAuthorization(CardNumber, Pin,
amount)
           "Authorize Cash Withdrawal"("ATM card #", PIN, Amount) : Bank -> env_l
           (* Request the Bank to authorize the cash withdrawal. *)
           Bank.RequestWithdrawalAuthorization returns
        Transaction.HandleAmountEntry returns

        "Withdrawal Accepted" (Amount) : env_l -> Bank
        (* Bank allows the cash withdrawal. *)

        Bank invokes Transaction.HandleBankReply(result)
           Transaction invokes Cash_Dispenser.DispenseCash(amount)
           (* Dispense the cash to the user. *)
           "Cash" : Cash_Dispenser -> env_l
           Cash_Dispenser.DispenseCash returns
```

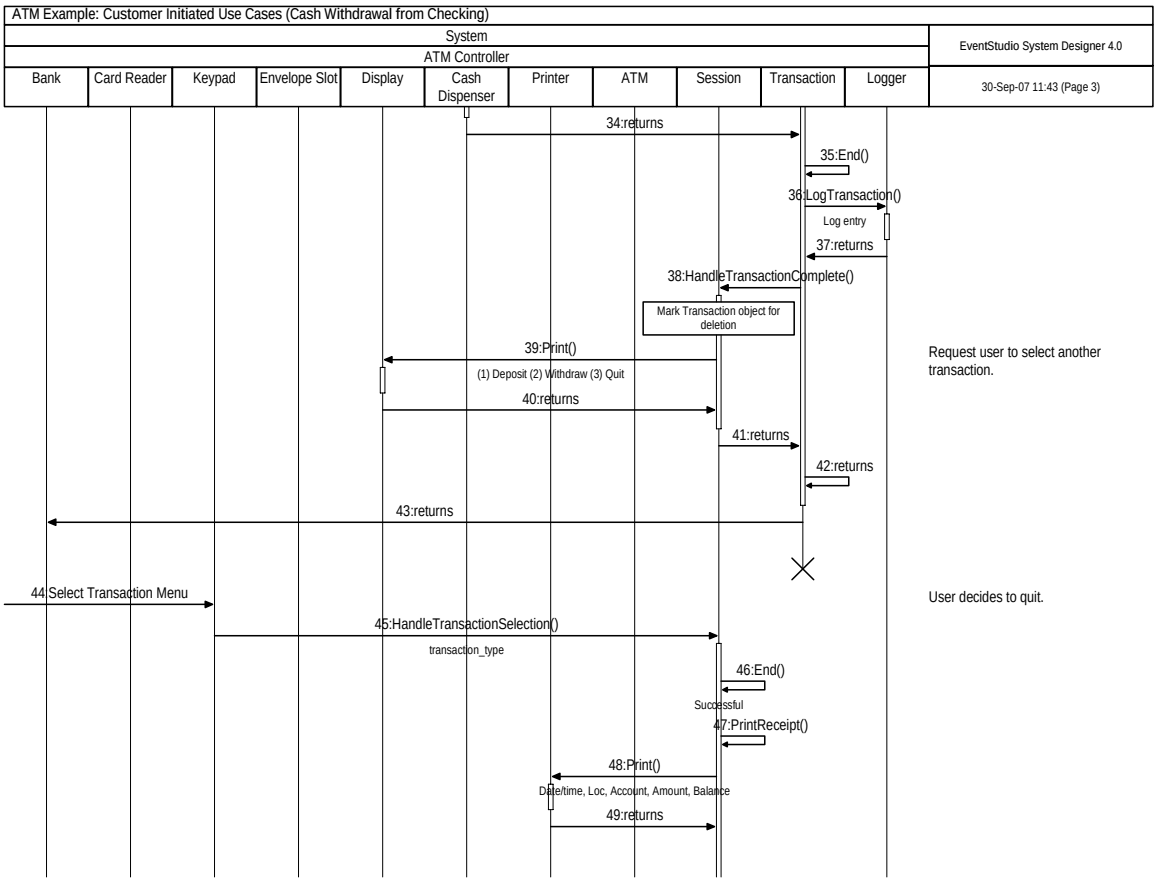
Method invoke and return are modeled with the "invokes" and "returns" statements.

© Copyright EventHelix.com Inc. All rights reserved.

```
      leg "Cash Deposit into Checking Account":
        Session creates Transaction
        (* The Session creates the DepositCashTransaction object. *)

        Session invokes Transaction.Start
          Transaction invokes Display.Print("Enter the amount you wish to deposit")
          (* Request the customer to enter the amount to be withdrawn. *)
          Display.Print returns
        Transaction.Start returns
        Session.HandleTransactionSelection returns

        "Dollar Amount" : env_l -> Keypad
        Keypad invokes Transaction.HandleAmountEntry(amount)
        (* The customer keys in the dollar amount that he or she wishes to deposit. *)
           Transaction invokes Display.Print("Please insert the cash envelope")
           (* Request the customer to deposit the cash envelope. *)
           Display.Print returns
        Transaction.HandleAmountEntry returns

       "Envelope With Cash" : env_l -> Envelope_Slot
       (* Customer inserts the envelope. *)
       Envelope_Slot invokes Transaction.HandleEnvelopeReceived
          Transaction invokes Display.Print("Cash deposit successful")
          Display.Print returns

   endcase

           Transaction invokes Transaction.End
              Transaction invokes Logger.LogTransaction("Log entry")
              Logger.LogTransaction returns

              Transaction invokes Session.HandleTransactionComplete
                 Session takes action "Mark Transaction object for deletion"
                   Session invokes Display.Print("(1) Deposit (2) Withdraw (3) Quit")
                   (* Request user to select another transaction. *)
                   Display.Print returns

                 Session.HandleTransactionComplete returns
              Transaction.End returns

        if "Cash Withdrawal from Checking Account"
           Transaction.HandleBankReply returns
        endif

        if "Cash Deposit into Checking Account"
           Transaction.HandleEnvelopeReceived returns
        endif

        Transaction deletes Transaction

        "Select Transaction Menu" : env_l -> Keypad
       (* User decides to quit. *)
       Keypad invokes Session.HandleTransactionSelection(transaction_type)
          Session invokes Session.End(Successful)
             Session invokes Session.PrintReceipt
                Session invokes Printer.Print("Date/time, Loc, Account, Amount, Balance")
                Printer.Print returns
             Session.PrintReceipt returns
             Session invokes Card_Reader.EjectCard
             Card_Reader.EjectCard returns
             Session invokes ATM.HandleSessionComplete
               ATM takes action "Mark Session for Deletion"
             ATM.HandleSessionComplete returns
          Session.End returns
       Session.HandleTransactionSelection returns
       Session deletes Session

label exit:

endfeature
```

7. Click **Control-Q** to regenerate the sequence diagram. A page from the generated sequence diagram is shown below:

© Copyright EventHelix.com Inc. All rights reserved.



4.2 Class Roles and Responsibilities

The next step in the analysis process is to scan the use case diagrams to identify the roles and responsibilities of individual classes. We will use the following diagrams to aid in the analysis:

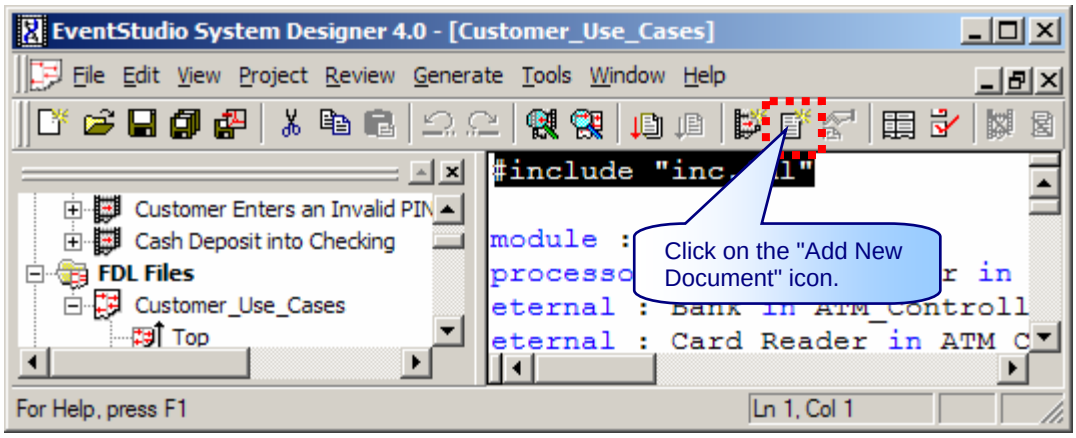
Interface Sequence Diagrams	A sequence diagram that just lists the interactions involving one module, processor or object.
Summary Document	Object wise summary of the interactions. The HTML document lists the interactions on a per object basis.
Interface Collaboration Diagrams	A collaboration diagram that just lists the interactions involving one module, processor or object.

4.2.1 Interface Sequence Diagram

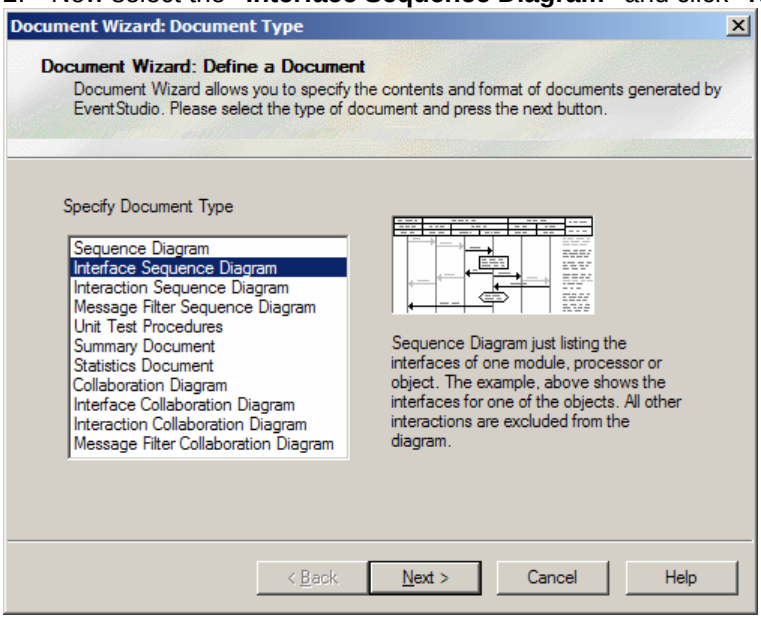
Use the interface sequence diagram to and role of important entities in the use case diagrams.
The following steps define the interface collaboration diagram:

1. Click on the **"Add Document"** icon (Shortcut **Control+D**).

© Copyright EventHelix.com Inc. All rights reserved.

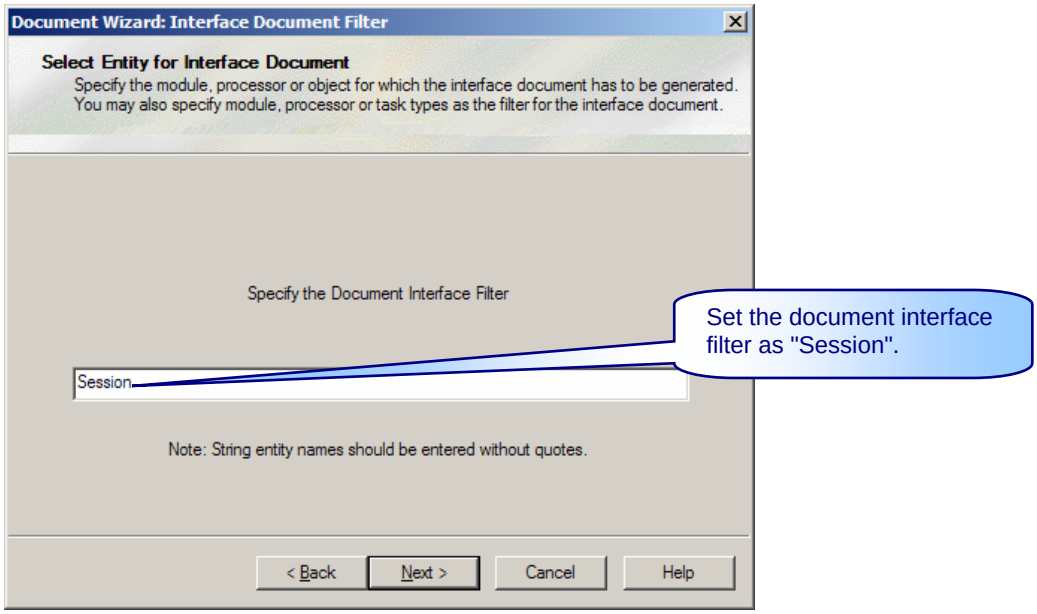


2. Now select the **"Interface Sequence Diagram"** and click **"Next"**.

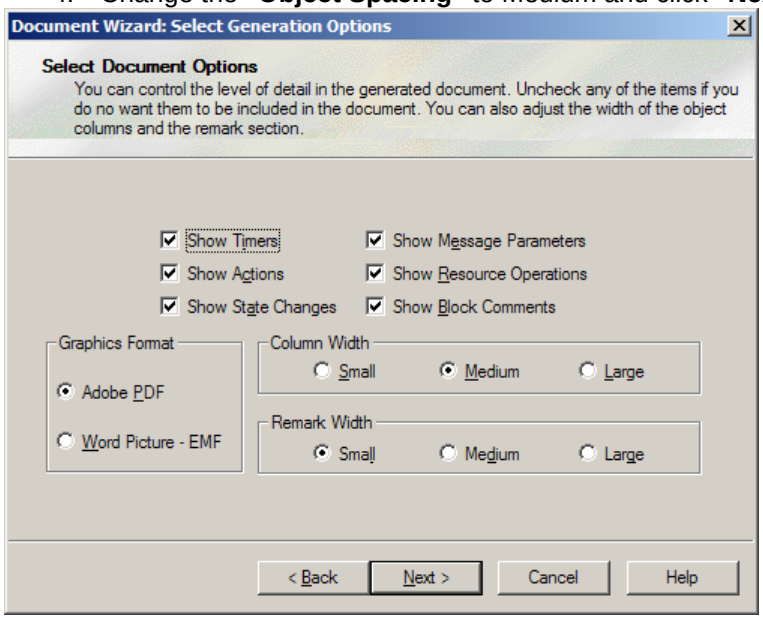


3. Set the **"Document Interface Filter"** as **"Session"** and click **"Next".**

© Copyright EventHelix.com Inc. All rights reserved.

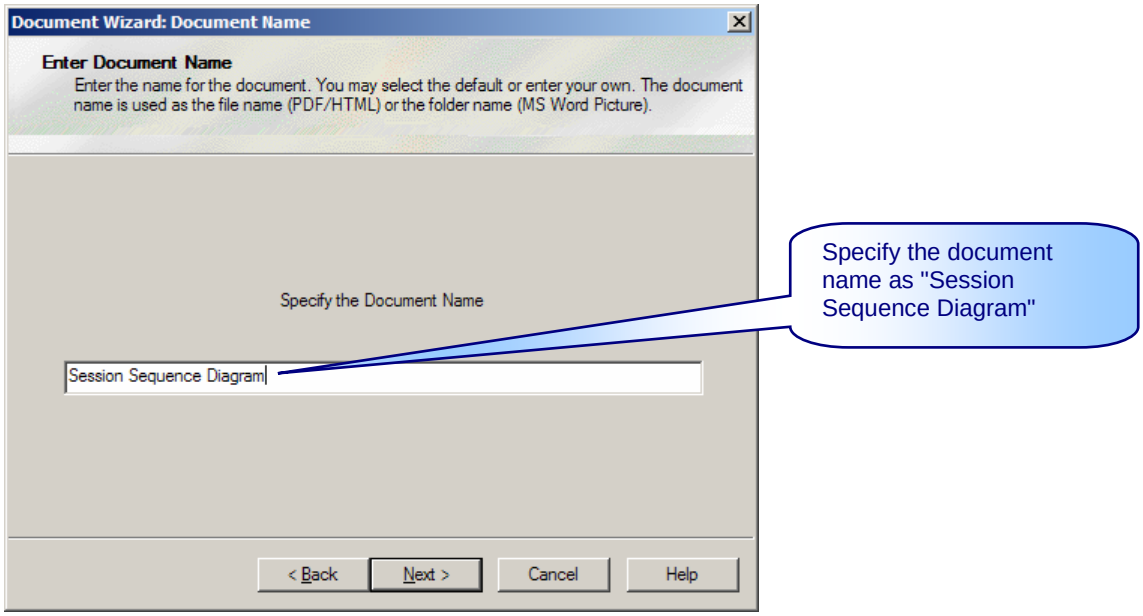


4. Change the **"Object Spacing"** to Medium and click **"Next"**.

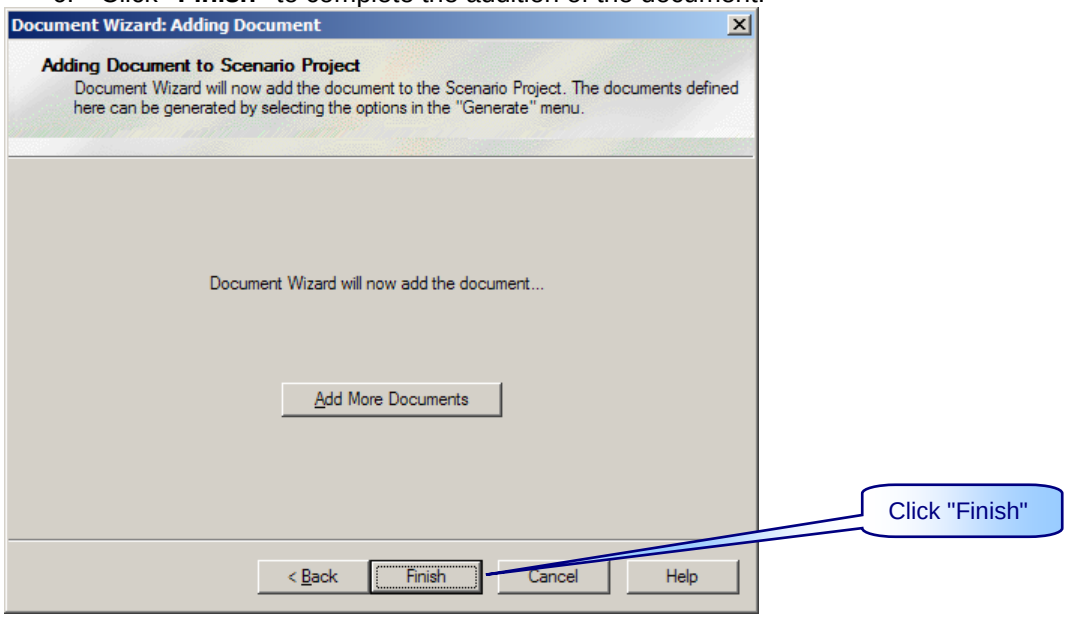


5. Enter the Document Name and click **"Next"**.

© Copyright EventHelix.com Inc. All rights reserved.

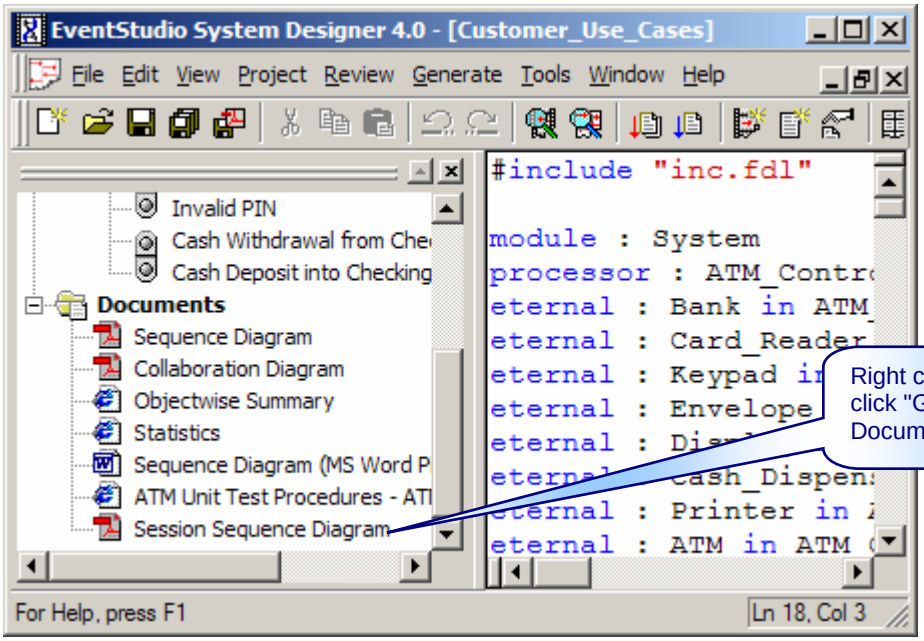


6. Click **"Finish"** to complete the addition of the document.

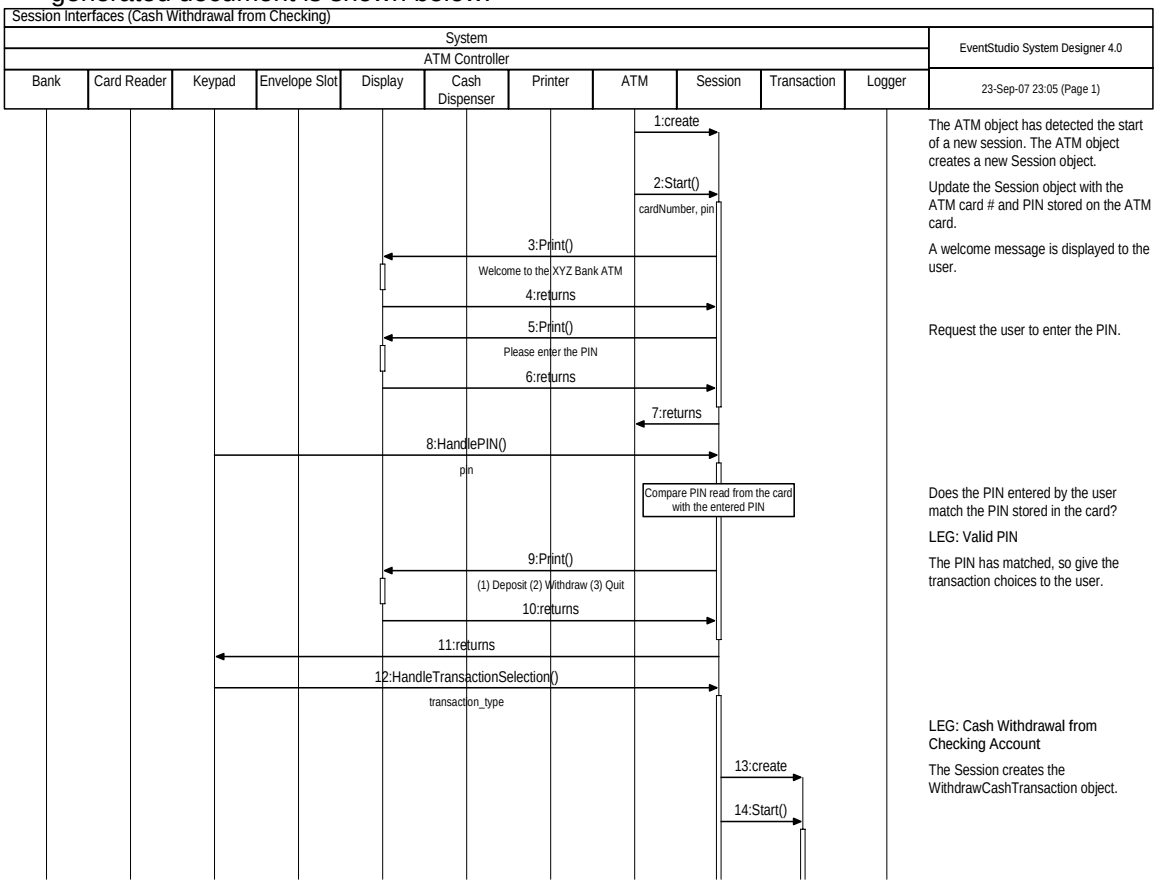


7. Now right click on the "Session Sequence Diagram" in the left pane and select "Generate Document" from the popup menu.

© Copyright EventHelix.com Inc. All rights reserved.



8. EventStudio generates the document and opens the document. One of the pages of the generated document is shown below.



© Copyright EventHelix.com Inc. All rights reserved.

4.2.2 Object Wise Summary

The object-wise summary document lists the interactions for each object. This aids in identifying the roles and responsibilities of individual classes. Generate this document by just right clicking on the "Objectwise Summary".

The first page of the generated document is shown below.

Objectwise Summary

EventStudio System Designer 4.0
Generated on 29-Sep-07 08:58

Cash Withdrawal from Checking

Legs Taken:

- Valid PIN
- Cash Withdrawal from Checking Account

Bank Actions

1. Transaction invokes Bank.RequestWithdrawalAuthorization() (CardNumber, Pin, amount)
2. Send Authorize Cash Withdrawal (ATM card #, PIN, Amount) to env_l
3. Bank.RequestWithdrawalAuthorization returns
4. Receive Withdrawal Accepted (Amount) from env_l
5. Bank invokes Transaction.HandleBankReply() (result)
6. Transaction.HandleBankReply returns

Card_Reader Actions

1. Receive Insert Credit Card from env_l
2. Card_Reader invokes ATM.HandleCreditCardSwipe() (cardNumber, pin)
3. ATM.HandleCreditCardSwipe returns
4. Session invokes Card_Reader.EjectCard()
5. Card_Reader.EjectCard returns

Keypad Actions

1. Receive Enter PIN from env_l
2. Keypad invokes Session.HandlePIN() (pin)
3. Session.HandlePIN returns
4. Receive Select Transaction Menu from env_l
5. Keypad invokes Session.HandleTransactionSelection() (transaction_type)
6. Session.HandleTransactionSelection returns
7. Receive Dollar Amount from env_l
8. Keypad invokes Transaction.HandleAmountEntry() (amount)
9. Transaction.HandleAmountEntry returns
10. Receive Select Transaction Menu from env_l
11. Keypad invokes Session.HandleTransactionSelection() (transaction_type)
12. Session.HandleTransactionSelection returns

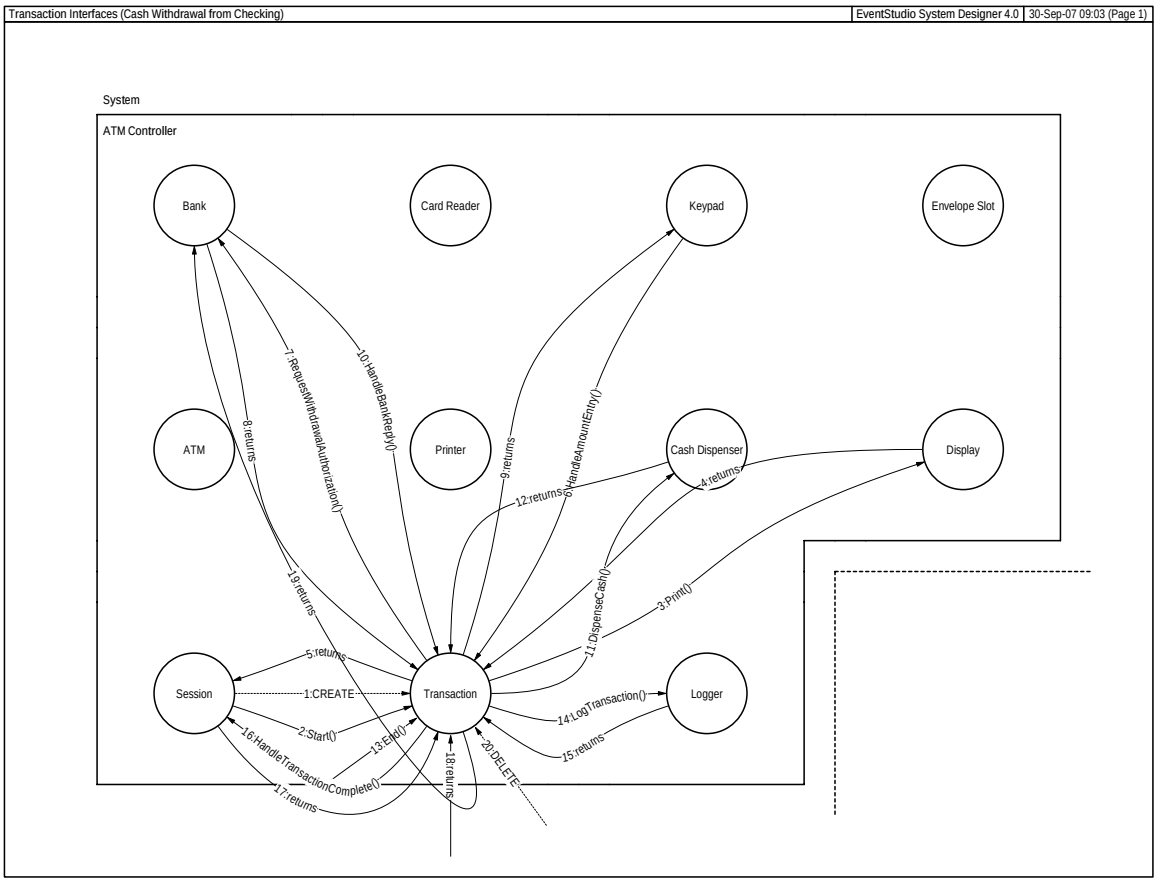
Display Actions

1. Session invokes Display.Print() (Welcome to the XYZ Bank ATM)
2. Display.Print returns
3. Session invokes Display.Print() (Please enter the PIN)
4. Display.Print returns
5. Session invokes Display.Print() ((1) Deposit (2) Withdraw (3) Quit)
6. Display.Print returns
7. Transaction invokes Display.Print() (Enter the Amount to be Withdrawn)
8. Display.Print returns
9. Session invokes Display.Print() ((1) Deposit (2) Withdraw (3) Quit)

© Copyright EventHelix.com Inc. All rights reserved.

10. Display.Print returns

4.2.3 Interface Collaboration Diagram

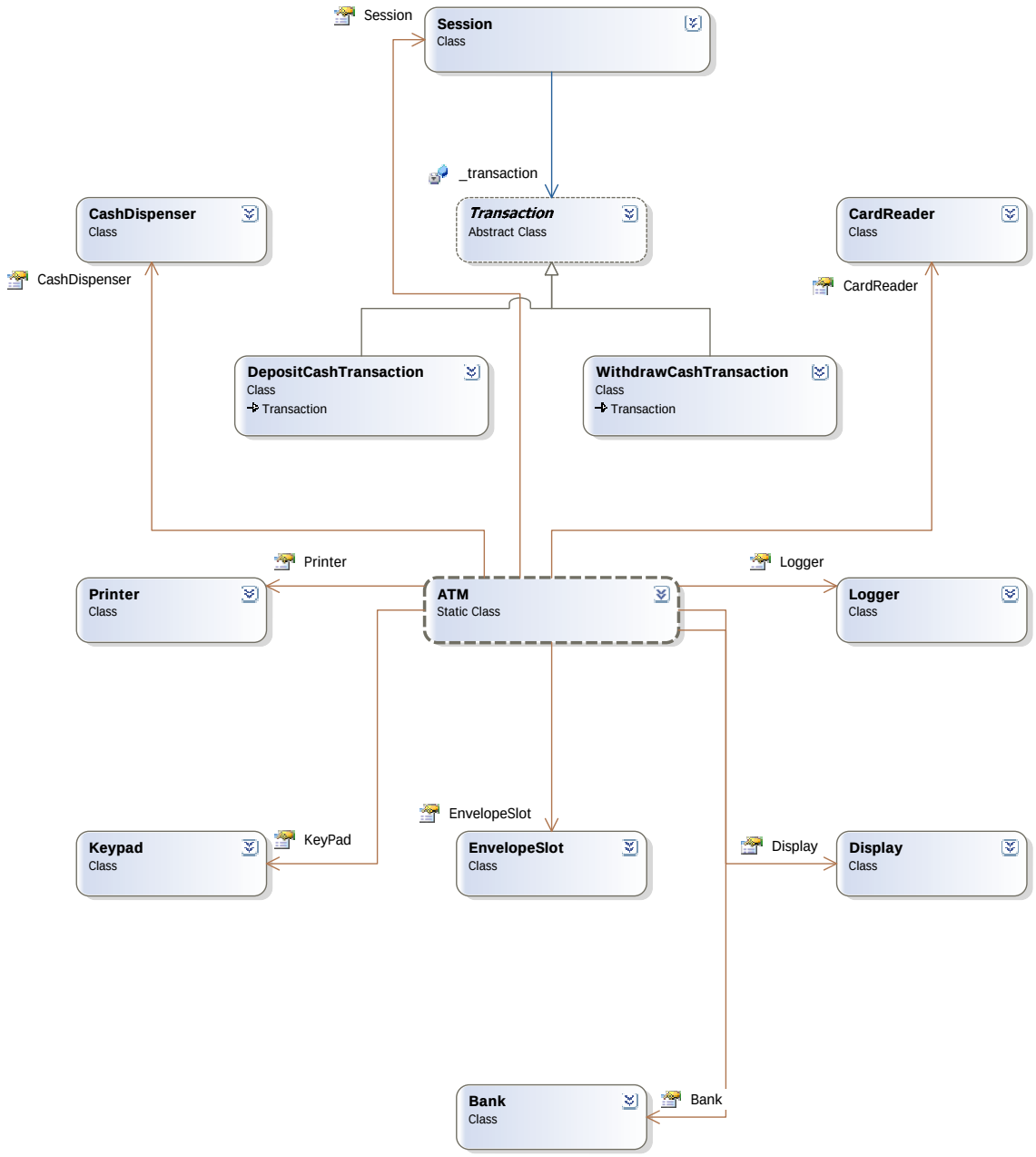


4.3 Class Diagrams

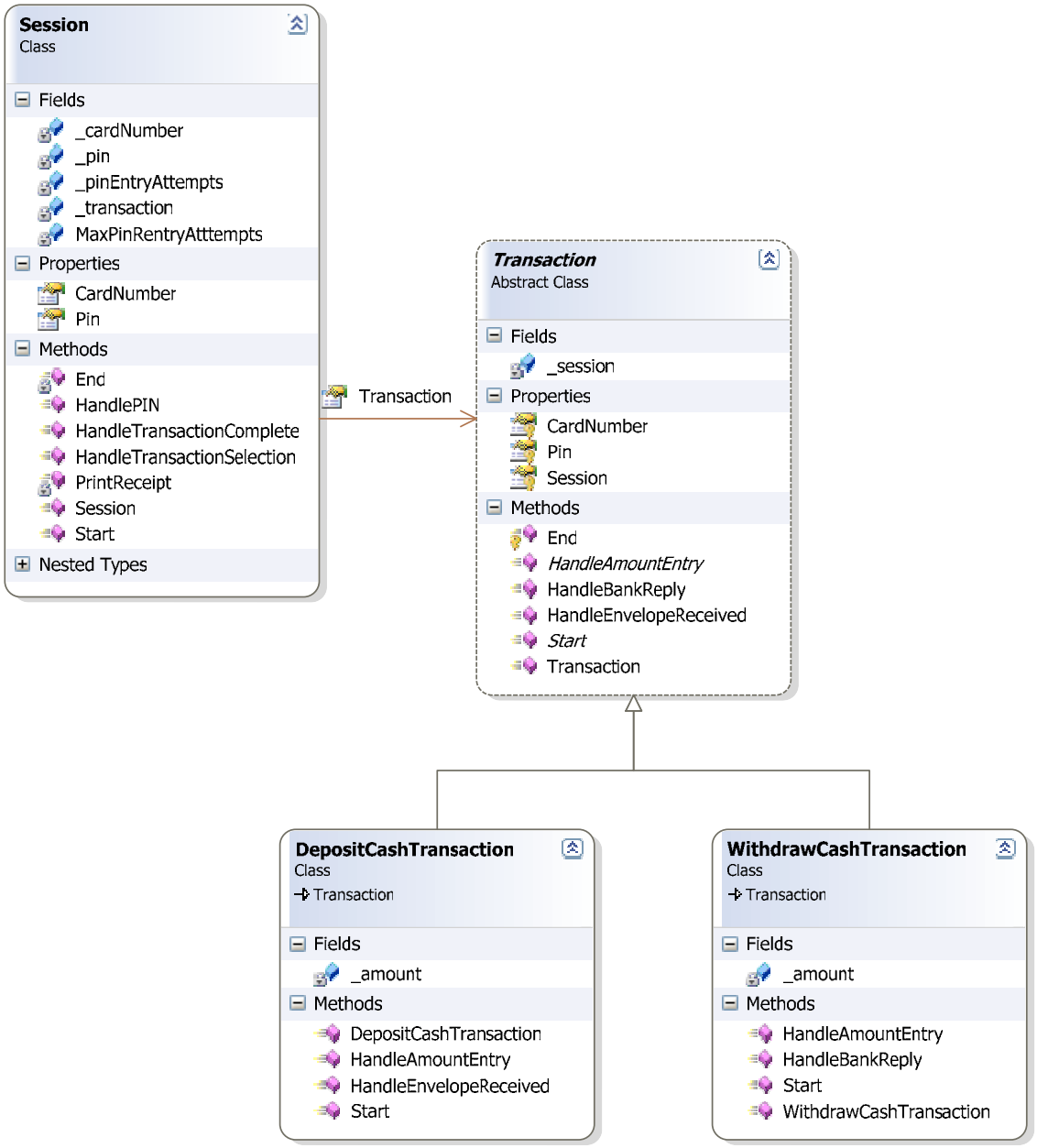
Now we have all the information for drawing the class diagrams for the classes implementing the ATM[2].

[2] We have generated these class diagrams with Visual Studio 2005. You may use any freeware of commercial UML tool to generate the class diagrams.

© Copyright EventHelix.com Inc. All rights reserved.



© Copyright EventHelix.com Inc. All rights reserved.



4.4 Skeleton Source Code in C#

The VisualStudio 2005 project containing the skeleton source code for the project can be downloaded from:
http://www.EventHelix.com/EventStudio/Tutorial/Use_Case_Sequence_Diagram_Tutorial.zip

The ZIP file also contains the Use Case and Detailed Design Scenario Projects.

© Copyright EventHelix.com Inc. All rights reserved.