APPENDIX I

This appendix supports chapter 4 and presents the most relevant parts of the source code (C#) of the prototype, which has been developed in order to validate the proposal of the solution.

The source code in this appendix includes the main algorithms which have been developed as well as all the main classes which have been created in accordance with the contents in chapter 4.

Process Loading Risk Profiles

```
private void LoadRiskProfiles()
{
        RiskProfileList profileList = RiskProfileManager.ReadList();
        for (int ctRp = 0; ctRp < profileList.Count; ctRp++)
        {
                // Load the profileList with their associated events
                profileList[ctRp].RiskProfile_EventList =
                   RiskProfileManager.GetEvents(profileList[ctRp].RiskProfile_Id);
                for (int ctRpEvent = 0; ctRpEvent <
                   profileList[ctRp].RiskProfile_EventList.Count; ctRpEvent++)
                {
                        // Get the parent id's list of each event
                        profileList[ctRp].RiskProfile_EventList[ctRpEvent].PidList =
                           RiskProfileEventManager.GetParentEventsList
                           (profileList[ctRp].RiskProfile_EventList[ctRpEvent].RiskProfileEven
                           t_Id);
                        // Get the list of triggers related with each event
                        profileList[ctRp].RiskProfile_EventList[ctRpEvent].relatedTriggerList
                           = RiskProfileEventManager.GetRelatedTriggerList
                           (profileList[ctRp].RiskProfile_EventList[ctRpEvent].RiskProfileEven
                           t_Id);
                        // Get the list of the user role allowed for each event
                        profileList[ctRp].RiskProfile_EventList[ctRpEvent].RiskProfileEvent_Us
                           erRoleAllowed =
                           RiskProfileEventManager.GetUserRoleAllowedList(profileList[ctRp].Ri
                           skProfile_EventList[ctRpEvent].RiskProfileEvent_Id);
                }
        }
}
```

Process Loading Sequences of Related Operations

```
private void LoadSequenceOfTriggerOccurrences()
{
        TriggerOccurrenceList tgoList =
           TriggerOccurrenceManager.ReadList(Program.lastIdOfTriggerOccurrence);
        if (tgoList != null)
        {
                bool inserted = false;
                // For each trigger occurrence
                for (int ctTgrOcc = 0; ctTgrOcc < tgoList.Count; ctTgrOcc++)
                {
                inserted = false;
                // if the occurred trigger is an insert trigger: verify trigger parent info;
                    else verify trigger occurrence info only
                if (TriggerManager.Trigger
                    (tgoList[ctTgrOcc].TriggerOccurrence_TriggerId).Trigger_Type == 1 &&
                    ParentInfoManager.GetParentInfoList(tgoList[ctTgrOcc].TriggerOccurrence_
                    Id) != null)
                {
                        // see if there is a list which contains the parent's document: if
                           yes- insert this occurrence in this list;
                        for (int ctS = 0; ctS < sequenceList.Count; ctS++)
                        {
                                if (!inserted)
                                        for (int ctTgo = 0; ctTgo <
                                            sequenceList[ctS].Sequence_tgoList.Count; ctTgo++)
                                        {
                                             if (ParentInfoManager.GetParentDocuments
                                                  (tgoList[ctTgrOcc].TriggerOccurrence_Id).Con
                                                  tainsDoc(TriggerOccurrenceInfoManager.GetDoc
                                                  umentInfo(sequenceList[ctS].Sequence_tgoList
                                                  [ctTgo].TriggerOccurrence_Id)))
                                             {
                                                       sequenceList[ctS].Sequence_tgoList.Add(
                                                           tgoList[ctTgrOcc]);
                                                       sequenceList[ctS].Sequence_possibleRPLi
                                                           st = null;
                                                       inserted = true;
                                                       break;
                                               }
                                        }
                                }
                        }
                        if (!inserted)
```

```
                    {
                        // see if there is a list which contains trigger occurrences
                            related to the same document: if yes - insert this trigger
                            occurrence in this list; else this trigger occurrence is
                            inserted in a new list
                        for (int ctS2 = 0; ctS2 < sequenceList.Count; ctS2++)
                        {
                                if (!inserted)
                                for (int ctTgo2 = 0; ctTgo2 <
                                    sequenceList[ctS2].Sequence_tgoList.Count;
                                    ctTgo2++)
                                {
                                    if
                                        (TriggerOccurrenceInfoManager.GetDocument
                                        Info
                                        (sequenceList[ctS2].Sequence_tgoList[ctTg
                                        o2].TriggerOccurrence_Id).IsEqual(Trigger
                                        OccurrenceInfoManager.GetDocumentInfo(tgo
                                        List[ctTgrOcc].TriggerOccurrence_Id)))
                                    {
                                      sequenceList[ctS2].Sequence_tgoList.Add
                                            (tgoList[ctTgrOcc]);
                                      sequenceList[ctS2].Sequence_possibleRPL
                                      ist = null;
                                      inserted = true;
                                      break;
                                    }
                                }
                            }
                        }
                        if (!inserted)
                        {
                            TriggerOccurrenceList l = new TriggerOccurrenceList();
                            l.Add(tgoList[ctTgrOcc]);
                            Sequence seq = new Sequence(l);
                            seq.Sequence_possibleRPList = null;
                            sequenceList.Add(seq);
                        }
                    }
                    Program.lastIdOfTriggerOccurrence = tgoList[tgoList.Count -
                          1].TriggerOccurrence_Id;
                }
            }
```

Process Classification of Sequences

```
private void ClassificationOfSequences()
{
        int i = 0, ctEvt = 0, ctSeq = 0, ctTgr = 0, ctRp = 0;
        PossibleRiskProfileList rpl;
        for (ctSeq = 0; ctSeq < sequenceList.Count; ctSeq++)
        {
        sequenceList[ctSeq].Sequence_possibleRPList = null;
        rpl = new PossibleRiskProfileList();
        for (i = 0; i < sequenceList[ctSeq].Sequence_tgoList.Count; i++)
        {
                if (i == 0)
                {
                        //see if there is, at least, for each sequence, one risk profile that
                            begins with the same event
                        // if yes, insert in the possibleRPList of this sequence the list of
                            the risk profile that the sequence can be classified
                        // if none risk profile identified, possibleRPList = null
                         for (ctRp = 0; ctRp < profileList.Count; ctRp++)
                        {
                                for (ctEvt = 0; ctEvt <
                                    profileList[ctRp].RiskProfile_EventList.Count; ctEvt++)
                                {
                                    bool found = false;
                                    if(profileList[ctRp].RiskProfile_EventList[ctEvt].PidL
                                        ist == null)
                                       for (ctTgr = 0; ctTgr <
                                              profileList[ctRp].RiskProfile_EventList[
                                              ctEvt].relatedTriggerList.Count;
                                              ctTgr++)
                                        {
                                              if
                                                   (profileList[ctRp].RiskProfile_Even
                                                   tList[ctEvt].relatedTriggerList[ctT
                                                   gr].Trigger_Id ==
                                                   sequenceList[ctSeq].Sequence_tgoLis
                                                   t[i].TriggerOccurrence_TriggerId)
                                               {
                                                  PossibleRiskProfile prp = new
                                                       PossibleRiskProfile();
                                                  prp.RiskProfile = new
                                                  RiskProfile(profileList[ctRp]);
                                                  if
                                                       (profileList[ctRp].RiskProfi
                                                       le_EventList.Count == 1)
                                                           prp.Status = 1;
```

```
                                        else
                                                prp.Status = 2;
                                        found = true;
                                       if
                                            (profileList[ctRp].RiskProfi
                                            le_EventList[ctEvt].RiskProf
                                            ileEvent_UserRoleAllowed.Con
                                            tains(sequenceList[ctSeq].Se
                                            quence_tgoList[i].TriggerOcc
                                            urrence_User))
                                                prp.RiskProfile.RiskProf
                                                     ile_EventList[ctEvt
                                                     ].AuditUser = 1;
                                        else
                                                prp.RiskProfile.RiskProf
                                                     ile_EventList[ctEvt
                                                     ].AuditUser = -1;
                                       prp.RiskProfile.RiskProfile_Even
                                            tList[ctEvt].AuditTime = 0;
                                       rpl.Add(prp);
                                       found = true;
                                    }
                                }
                                if (found)
                                        break;
                            }
                        }
                        sequenceList[ctSeq].Sequence_possibleRPList = rpl;
                        sequenceList[ctSeq].Sequence_tgoList[i].AuditOrder = 1;
                    }
                }
            }
            // see if the event follows the defined order
            PossibleRiskProfileList[] rplTemp = new
                  PossibleRiskProfileList[sequenceList.Count];
            PossibleRiskProfileList[] nrplTemp = new
                  PossibleRiskProfileList[sequenceList.Count];
            i = 0; ctEvt = 0; ctSeq = 0; ctTgr = 0; ctRp = 0;
            bool processed;
            for (ctSeq = 0; ctSeq < sequenceList.Count; ctSeq++)
            {
                rplTemp[ctSeq] = new PossibleRiskProfileList();
                nrplTemp[ctSeq] = new PossibleRiskProfileList();
                processed = false;
                for (i = 0; i < sequenceList[ctSeq].Sequence_tgoList.Count; i++)
                {
                    if (i != 0)
                    {
```

```
                            processed = true;
                            //At the final, the sequences which have not possibleRP, are not
recognized in the risk profile repository
                            // verify if occurrence (i) exists in the events list of the
possibleRPList
                            for (ctRp = 0; ctRp <
                                 sequenceList[ctSeq].Sequence_possibleRPList.Count; ctRp++)
                            {
                                int order = 0;  // 0 - at beginning; 1 - validated order; -1
                                    - not validated order
                                 int countEvt = 0;
                                 for (ctEvt = 0; ctEvt <
                                    sequenceList[ctSeq].Sequence_possibleRPList[ctRp].RiskPr
                                    ofile.RiskProfile_EventList.Count; ctEvt++)
                                 {
                                      for (ctTgr = 0; ctTgr <
                                         sequenceList[ctSeq].Sequence_possibleRPList[ctRp].Ri
                                         skProfile.RiskProfile_EventList[ctEvt].relatedTrigge
                                         rList.Count; ctTgr++)
                                     {
                                             if
                                                 (sequenceList[ctSeq].Sequence_possibleRPLis
                                                 t[ctRp].RiskProfile.RiskProfile_EventList[c
                                                 tEvt].relatedTriggerList[ctTgr].Trigger_Id
                                                 ==
                                                 sequenceList[ctSeq].Sequence_tgoList[i].Tri
                                                 ggerOccurrence_TriggerId)
                                             {
                                                  int[] aud = new int[2];
                                                 aud = ParentsVerification(i,
                                                    sequenceList[ctSeq].Sequence_tgoLis
                                                    t,
                                                    sequenceList[ctSeq].Sequence_possib
                                                    leRPList[ctRp].RiskProfile.RiskProf
                                                    ile_EventList[ctEvt].PidList,
                                                    sequenceList[ctSeq].Sequence_possib
                                                    leRPList[ctRp].RiskProfile.RiskProf
                                                    ile_EventList[ctEvt]);
                                                  if (aud != null && aud[0] == 1)
                                                  {
                                                     if (order == 0)
                                                             order = 1;
                                                      sequenceList[ctSeq].Sequence_po
                                                         ssibleRPList[ctRp].RiskProfi
                                                         le.RiskProfile_EventList[ctE
                                                         vt].AuditTime = aud[1];
                                                      sequenceList[ctSeq].Sequence_tg
                                                         oList[i].AuditOrder = 1;
```

```
                        if
                           (sequenceList[ctSeq].Sequenc
                           e_possibleRPList[ctRp].RiskP
                           rofile.RiskProfile_EventList
                           [ctEvt].RiskProfileEvent_Use
                           rRoleAllowed.Contains(sequen
                           ceList[ctSeq].Sequence_tgoLi
                           st[i].TriggerOccurrence_User
                           ))
                                sequenceList[ctSeq].Sequ
                                   ence_possibleRPList[c
                                   tRp].RiskProfile.Risk
                                   Profile_EventList[ctE
                                   vt].AuditUser = 1;
                        else
                                sequenceList[ctSeq].Sequ
                                   ence_possibleRPList[c
                                   tRp].RiskProfile.Risk
                                   Profile_EventList[ctE
                                   vt].AuditUser = -1;
    }
    else
    {
       order = -1;
       nrplTemp[ctSeq].Add(sequenceList[ctSeq].Sequ
            ence_possibleRPList[ctRp]);
    }

    break;
}
else
    countEvt++;
if ((i == sequenceList[ctSeq].Sequence_tgoList.Count
      - 1) &&
      (sequenceList[ctSeq].Sequence_possibleRPList[c
      tRp].RiskProfile.RiskProfile_EventList.Count >
      i + 1))
{
    int[] aud = new int[2];
    aud = ParentsVerificationFirstLast(i,
       sequenceList[ctSeq].Sequence_tgoList,
       sequenceList[ctSeq].Sequence_possibleRPList[
       ctRp].RiskProfile.RiskProfile_EventList[i +
       1].PidList,sequenceList[ctSeq].Sequence_poss
       ibleRPList[ctRp].RiskProfile.RiskProfile_Eve
       ntList[i + 1]);
    if (aud[1] == 1)
```

```
                    sequenceList[ctSeq].Sequence_possibleRPList
                       [ctRp].RiskProfile.RiskProfile_EventList
                       [i + 1].AuditTime = 1;
                else
                    sequenceList[ctSeq].Sequence_possibleRPList
                       [ctRp].RiskProfile.RiskProfile_EventList
                       [i + 1].AuditTime = -1;
            }
        }
        if (order == 1)
        {
            if(!rplTemp[ctSeq].Contains(sequenceList[ctSeq].Sequ
               ence_possibleRPList[ctRp]))
                    rplTemp[ctSeq].Add(sequenceList[ctSeq].Sequ
                       ence_possibleRPList[ctRp]);
            break;
        }
        else
             if (order == 0 && countEvt >=
                sequenceList[ctSeq].Sequence_possibleRPList[ctR
                p].RiskProfile.RiskProfile_EventList.Count)
                   nrplTemp[ctSeq].Add(sequenceList[ctSeq].Sequ
                      ence_possibleRPList[ctRp]);
    }
    if (order != -1)  // Status: 1-complete; 2-incomplete;
    {
        int nAuditedEvts =
            NumberOfAuditedEvents(sequenceList[ctSeq].Sequence_t
            goList);
        if (nAuditedEvts ==
            sequenceList[ctSeq].Sequence_tgoList.Count)
       {
            if (nAuditedEvts ==
                sequenceList[ctSeq].Sequence_possibleRPList[ctR
                p].RiskProfile.RiskProfile_EventList.Count)
                    sequenceList[ctSeq].Sequence_possibleRPList
                       [ctRp].Status = 1;
           else
                    if (nAuditedEvts <
                       sequenceList[ctSeq].Sequence_possibleRPL
                       ist[ctRp].RiskProfile.RiskProfile_EventL
                       ist.Count)
                            sequenceList[ctSeq].Sequence_possib
                               leRPList[ctRp].Status = 2;
        }
        else
        {
```

```
                    if (nAuditedEvts <
                         sequenceList[ctSeq].Sequence_tgoList.Count)
                                if (nAuditedEvts ==
                                   sequenceList[ctSeq].Sequence_possible
                                   RPList[ctRp].RiskProfile.RiskProfile_
                                   EventList.Count)
                                          sequenceList[ctSeq].Sequence_p
                                             ossibleRPList[ctRp].Status
                                             = 3;
                                else
                                          if (nAuditedEvts <
                                             sequenceList[ctSeq].Sequenc
                                             e_possibleRPList[ctRp].Risk
                                             Profile.RiskProfile_EventLi
                                             st.Count)
                                               sequenceList[ctSeq].Se
                                                  quence_possibleRPLi
                                                  st[ctRp].Status =
                                                  4;
                }
            }
            else
            {
                if (order == -1)
                     if(rplTemp[ctSeq].Contains(sequenceList[ctSeq].Seq
                        uence_possibleRPList[ctRp]))
                           rplTemp[ctSeq].Remove(sequenceList[ctSeq]
                              .Sequence_possibleRPList[ctRp]);
            }
        }
    }
    else
        for (int s = 0;
           s<sequenceList[ctSeq].Sequence_possibleRPList.Count; s++)
                 if ((sequenceList[ctSeq].Sequence_tgoList.Count == 1)
                     &&
                     (sequenceList[ctSeq].Sequence_possibleRPList[s].R
                     iskProfile.RiskProfile_EventList.Count > 1))
                 {
                       int[] aud = new int[2];
                        aud = ParentsVerificationFirstLast(i,
                             sequenceList[ctSeq].Sequence_tgoList,
                             sequenceList[ctSeq].Sequence_possibleRPLi
                             st[s].RiskProfile.RiskProfile_EventList[i
                             + 1].PidList,
                             sequenceList[ctSeq].Sequence_possibleRPLi
                             st[s].RiskProfile.RiskProfile_EventList[i
                             + 1]);
```

```
                                    if (aud[1] == 1)
                                        sequenceList[ctSeq].Sequence_possibleRPList[
                                           s].RiskProfile.RiskProfile_EventList[i +
                                           1].AuditTime = 1;
                                    else
                                        sequenceList[ctSeq].Sequence_possibleRPList[
                                            s].RiskProfile.RiskProfile_EventList[i +
                                            1].AuditTime = -1;
                                }
                    }
                    if (!processed)
                        rplTemp[ctSeq] = sequenceList[ctSeq].Sequence_possibleRPList;
                }
                ctSeq = 0;
                for (ctSeq = 0; ctSeq < sequenceList.Count; ctSeq++)
                {
                    for (int ctNrpr = 0; ctNrpr < nrplTemp[ctSeq].Count; ctNrpr++)
                    {
                        if (rplTemp[ctSeq].Contains(nrplTemp[ctSeq][ctNrpr]))
                            rplTemp[ctSeq].Remove(nrplTemp[ctSeq][ctNrpr]);
                    }
                    sequenceList[ctSeq].Sequence_possibleRPList = rplTemp[ctSeq];
                }
}
```

Main Classes

Class Sequence

```
public class Sequence
    {
        private TriggerOccurrenceList tgoList;
        private PossibleRiskProfileList possibleRPList;

        /// <summary>
        /// Default Constructor of the Sequence Class
        /// </summary>
        public Sequence()
        {
            tgoList = null;
            possibleRPList = null;
        }

        /// <summary>
        /// Constructor of the Sequence Class
        /// </summary>
        /// <param name="tgoL">Trigger Occurrence List</param>
        /// <param name="rpL">Possible List of risk profile which this sequence can be
                    classified</param>
        public Sequence(TriggerOccurrenceList tgoL, PossibleRiskProfileList rpL)
        {
            tgoList = tgoL;
            possibleRPList = rpL;
        }

        /// <summary>
        /// Constructor of the Sequence Class
        /// </summary>
        /// <param name="tgoL">Trigger Occurrence List</param>
        public Sequence(TriggerOccurrenceList tgoL)
        {
            tgoList = tgoL;
            possibleRPList = null;
        }

        /// <summary>
        /// Get/set the trigger occurrence list
        /// </summary>
        public TriggerOccurrenceList Sequence_tgoList
        {
            get
            {
                return tgoList;
```

```
            }
            set
            {
                tgoList = value;
            }
        }

        /// <summary>
        /// Get/set the possible risk profile list which this sequence can be classified
        /// </summary>
        public PossibleRiskProfileList Sequence_possibleRPList
        {
            get
            {
                return possibleRPList;
            }
            set
            {
                possibleRPList = value;
            }
        }
    }
```

Class PossibleRiskProfile

```
public class PossibleRiskProfile
    {
        private RiskProfile rp;
        private int status;  // 1 - complete; 2 - incomplete;

        /// <summary>
        /// Constructor of PossibleRiskProfile object
        /// </summary>
        public PossibleRiskProfile()
        {
            rp = null;
            status = 0;
        }
        public PossibleRiskProfile(PossibleRiskProfile other)
        {
            rp = new RiskProfile(other.rp);
            status = other.status;
        }

        /// <summary>
        /// Constructor of PossibleRiskProfile object
        /// </summary>
        /// <param name="r">risk profile</param>
```

```
        /// <param name="s">status: 1 - complete; 2 - incomplete</param>
        public PossibleRiskProfile(RiskProfile r, int s)
        {
            rp = r;
            status = s;
        }

        public RiskProfile RiskProfile
        {
            get
            {
                return rp;
            }
            set
            {
                rp = value;
            }
        }

        public int Status
        {
            get
            {
                return status;
            }
            set
            {
                status = value;
            }
        }
    }
```

Class RiskProfile

```
public class RiskProfile
    {
        private int id;
        private string description;
        private string transaction;
        private int type;
        private RiskProfileEventList events;

        /// <summary>
        /// Constructor of RiskProfile object
        /// </summary>
        public RiskProfile()
        {
            id = -1;
```

```
    description = string.Empty;
    transaction = string.Empty;
    type = -1;
}

public RiskProfile(RiskProfile other)
{
    id = other.id;
    description = other.description;
    transaction = other.transaction;
    type = other.type;
    if (other.events == null)
        events = null;
    else
    {
        events = new RiskProfileEventList();
        for (int i = 0; i < other.events.Count; i++)
        {
            RiskProfileEvent rpEvt = new RiskProfileEvent(other.events[i]);
            events.Add(rpEvt);
        }
    }

}

/// <summary>
/// Constructor of RiskProfile object
/// </summary>
/// <param name="i">risk profile id</param>
/// <param name="desc">risk profile description</param>
/// <param name="trans">risk profile transaction</param>
/// <param name="tp">risk profile type: 0-negative; 1-positive</param>
public RiskProfile(int i, string desc, string trans, int tp)
{
    id = i;
    description = desc;
    transaction = trans;
    type = tp;
}

/// <summary>
/// Constructor of RiskProfile object
/// </summary>
/// <param name="i">risk profile id</param>
/// <param name="desc">risk profile description</param>
/// <param name="trans">transaction of risk profile</param>
/// <param name="tp">risk profile type</param>
/// <param name="l">list of events of this risk profile</param>
```

```
public RiskProfile(int i, string desc, string trans, int tp, RiskProfileEventList l)
{
    id = i;
    description = desc;
    transaction = trans;
    type = tp;
    events = l;
}

/// <summary>
/// Get/set the risk profile id
/// </summary>
public int RiskProfile_Id
{
    get
    {
        return id;
    }
    set
    {
        id = value;
    }
}

/// <summary>
/// Get/set the risk profile description
/// </summary>
public string RiskProfile_Description
{
    get
    {
        return description;
    }
    set
    {
        description = value;
    }
}

/// <summary>
/// Get/set the risk profile transaction
/// </summary>
public string RiskProfile_Transaction
{
    get
    {
        return transaction;
    }
```

```
            set
            {
                transaction = value;
            }
        }

        /// <summary>
        /// Get/set the risk profile type: 0-negative; 1-positive
        /// </summary>
        public int RiskProfile_Type
        {
            get
            {
                return type;
            }
            set
            {
                type = value;
            }
        }

        /// <summary>
        /// Get/set the list of events of this risk profile
        /// </summary>
        public RiskProfileEventList RiskProfile_EventList
        {
            get
            {
                return events;
            }
            set
            {
                events = value;
            }
        }
    }
```

Class RiskProfileEvent

```
public class RiskProfileEvent
    {
        private int id;
        private string description;
        private string type;
        private RiskProfileEventList pid;
        private TriggerList relatedTriggers;
        private List<int> userRoleAllowed;
```

```
private int auditUser;
private int auditTime;

/// <summary>
/// Constructor of RiskProfileEvent object
/// </summary>
public RiskProfileEvent()
{
    id = -1;
    description = string.Empty;
    type = string.Empty;
    pid = null;
    auditUser = 0;
    auditTime = 0;
}
public RiskProfileEvent(RiskProfileEvent other)
{
    id = other.id;
    description = other.description;
    type = other.type;
    if (other.pid == null)
        pid = null;
    else
    {
        pid = new RiskProfileEventList();
        for (int i = 0; i < other.pid.Count; i++)
        {
            RiskProfileEvent rpEvt = new RiskProfileEvent(other.pid[i]);
            pid.Add(rpEvt);
        }
    }
    if (other.relatedTriggers == null)
        relatedTriggers = null;
    else
    {
        relatedTriggers = new TriggerList();
        for (int i = 0; i < other.relatedTriggers.Count; i++)
        {
            Trigger t = new Trigger(other.relatedTriggers[i]);
            relatedTriggers.Add(t);
        }
    }
    if (other.userRoleAllowed == null)
        userRoleAllowed = null;
    else
    {
        userRoleAllowed = new List<int>();
        for (int i = 0; i < other.userRoleAllowed.Count; i++)
```

```
        {
            userRoleAllowed.Add(other.userRoleAllowed[i]);
        }

    }
    auditUser = other.auditUser;
    auditTime = other.auditTime;
}

/// <summary>
/// Constructor of RiskProfileEvent object
/// </summary>
/// <param name="i">risk profile event id</param>
/// <param name="desc">risk profile event description</param>
/// <param name="tp">risk profile event type</param>
/// <param name="pidList">List of risk profile event parents</param>
public RiskProfileEvent(int i, string desc, string tp, RiskProfileEventList pidList)
{
    id = i;
    description = desc;
    type = tp;
    pid = pidList;
    auditUser = 0;
    auditTime = 0;
}

/// <summary>
/// Get/set the risk profile event id
/// </summary>
public int RiskProfileEvent_Id
{
    get
    {
        return id;
    }
    set
    {
        id = value;
    }
}

/// <summary>
/// Get/set the risk profile event description
/// </summary>
public string RiskProfileEvent_Description
{
    get
    {
```

```
            return description;
        }
        set
        {
            description = value;
        }
    }

    /// <summary>
    /// Get/set the risk profile event type
    /// </summary>
    public string RiskProfileEvent_Type
    {
        get
        {
            return type;
        }
        set
        {
            type = value;
        }
    }

    /// <summary>
    /// Get/set the list of events' parents
    /// </summary>
    public RiskProfileEventList PidList
    {
        get
        {
            return pid;
        }
        set
        {
            pid = value;
        }
    }

    /// <summary>
    /// Get/set the list of related triggers
    /// </summary>
    public TriggerList relatedTriggerList
    {
        get
        {
            return relatedTriggers;
        }
        set
```

```
            {
                relatedTriggers = value;
            }
        }

        /// <summary>
        /// Get/set the user role id allowed to this event
        /// </summary>
        public List<int> RiskProfileEvent_UserRoleAllowed
        {
            get
            {
                return userRoleAllowed;
            }
            set
            {
                userRoleAllowed = value;
            }

        }
        /// <summary>
        /// Get/set the audit user bit: 0 - not audit; 1 - authorized user; -1 -
unauthorized user
        /// </summary>
        public int AuditUser
        {
            get
            {
                return auditUser;
            }
            set
            {
                auditUser = value;
            }
        }
        /// <summary>
        /// Get/set the audit time bit: 0 - not audit; 1 - in time; -1 - delay
        /// </summary>
        public int AuditTime
        {
            get
            {
                return auditTime;
            }
            set
            {
                auditTime = value;
            }
```

```
        }
    }
```

Class Trigger

```
public class Trigger
    {
        private int id;
        private string description;
        private int type;

        /// <summary>
        /// Constructor of Trigger object
        /// </summary>
        public Trigger()
        {
            id = -1;
            description = string.Empty;
            type = -1;
        }
        public Trigger(Trigger other)
        {
            id = other.id;
            description = other.description;
            type = other.type;
        }

        /// <summary>
        /// Contructor of the Trigger object
        /// </summary>
        /// <param name="tgr_id"> Trigger id</param>
        /// <param name="tgr_description">Trigger description</param>
        /// <param name="tgr_type">Trigger type: 1 - insert; 2 - update; 3 - delete</param>
        public Trigger(int tgr_id, string tgr_description, int tgr_type)
        {
            id = tgr_id;
            description = tgr_description;
            type = tgr_type;
        }

        /// <summary>
        /// Get/set the trigger id
        /// </summary>
        public int Trigger_Id
        {
            get
            {
                return id;
```

```
            }
            set
            {
                id = value;
            }
        }

        /// <summary>
        /// Get/set the trigger description
        /// </summary>
        public string Trigger_Description
        {
            get
            {
                return description;
            }
            set
            {
                description = value;
            }
        }

        /// <summary>
        /// Get/set the trigger type
        /// </summary>
        public int Trigger_Type
        {
            get
            {
                return type;
            }
            set
            {
                type = value;
            }
        }

    }
```

Class TriggerOccurrence

```
public class TriggerOccurrence
    {
        private int id;
        private int triggerId;
        private DateTime datetime;
        private string dbTable;
```

```
        private int user;
        private int auditOrder;
        private int auditUser;

        /// <summary>
        /// Constructor of TriggerOccurrence object
        /// </summary>
        public TriggerOccurrence()
        {
            id = -1;
            triggerId = -1;
            datetime = DateTime.MaxValue;
            dbTable = string.Empty;
            user = -1;
            auditOrder = -1;
            auditUser = -1;
        }

        /// <summary>
        /// Constructor of the Trigger Occurrence object
        /// </summary>
        /// <param name="tgo_id">Trigger occurrence id</param>
        /// <param name="tgotgr_id">Trigger id associated to the trigger occurrence</param>
        /// <param name="tgo_datetime">Date and time of the trigger occurrence</param>
        /// <param name="tgo_dbtable">Database table associated to the trigger
occurrence</param>
        /// <param name="tgousr_id">User id responsible by the trigger occurrence</param>
        public TriggerOccurrence (int tgo_id, int tgotgr_id, DateTime tgo_datetime, string
               tgo_dbtable, int tgousr_id, int order, int user)
        {
            id = tgo_id;
            triggerId = tgotgr_id;
            datetime = tgo_datetime;
            dbTable = tgo_dbtable;
            user = tgousr_id;
            auditOrder = order;
            auditUser = user;
        }

        /// <summary>
        /// Get/set the trigger occurrence id
        /// </summary>
        public int TriggerOccurrence_Id
        {
            get
            {
                return id;
            }
```

```
        set
        {
            id = value;
        }
    }

    /// <summary>
    /// Get/set the trigger associated with the occurrence
    /// </summary>
    public int TriggerOccurrence_TriggerId
    {
        get
        {
            return triggerId;
        }
        set
        {
            triggerId = value;
        }
    }

    /// <summary>
    /// Get/set the date and time of the trigger occurrence
    /// </summary>
    public DateTime TriggerOccurrence_DateTime
    {
        get
        {
            return datetime;
        }
        set
        {
            datetime = value;
        }
    }

    /// <summary>
    /// Get/set the database tabel associated with the trigger occurrence
    /// </summary>
    public string TriggerOccurrence_dbTable
    {
        get
        {
            return dbTable;
        }
        set
        {
            dbTable = value;
```

```
            }
        }

        /// <summary>
        /// Get/set the user responsible by the trigger occurrence
        /// </summary>
        public int TriggerOccurrence_User
        {
            get
            {
                return user;
            }
            set
            {
                user = value;
            }
        }

        /// <summary>
        /// Get/set the audit order value: -1 do not audited; 0 do not pass; 1 pass
        /// </summary>
        public int AuditOrder
        {
            get
            {
                return auditOrder;
            }
            set
            {
                auditOrder = value;
            }
        }

        /// <summary>
        /// Get/set the audit user value: -1 do not audited; 0 user does not allowed; 1 user
allowed
        /// </summary>
        public int AuditUser
        {
            get
            {
                return auditUser;
            }
            set
            {
                auditUser = value;
            }
        }
```

```
}
```

Class User

```
public class User
{
    private int id;
    private string description;
    private int role;

    /// <summary>
    /// Constructor of User object
    /// </summary>
    public User()
    {
        id = -1;
        description = string.Empty;
        role = -1;
    }

    /// <summary>
    /// Contructor of the User object
    /// </summary>
    /// <param name="usr_id">User id</param>
    /// <param name="usr_description">User description</param>
    /// <param name="usr_role">User role: 1 - administrator</param>
    public User(int usr_id, string usr_description, int usr_role)
    {
        id = usr_id;
        description = usr_description;
        role = usr_role;
    }

    /// <summary>
    /// Get/set the user id
    /// </summary>
    public int User_Id
    {
        get
        {
            return id;
        }
        set
        {
            id = value;
        }
    }
```

```
        /// <summary>
        /// Get/set the user description
        /// </summary>
        public string User_Description
        {
            get
            {
                return description;
            }
            set
            {
                description = value;
            }
        }

        /// <summary>
        /// Get/set the user role
        /// </summary>
        public int User_Role
        {
            get
            {
                return role;
            }
            set
            {
                role = value;
            }
        }

    }
```

Class TriggerOccurrenceInfo

```
public class TriggerOccurrenceInfo
    {
        private int id;
        private int triggerOccurrenceId;
        private string attribute;
        private string val;
        private string description;

        /// <summary>
        /// Constructor of TriggerOccurrenceInfo object
        /// </summary>
```

```
public TriggerOccurrenceInfo()
{
    id = -1;
    triggerOccurrenceId = -1;
    attribute = string.Empty;
    val = string.Empty;
    description = string.Empty;
}

/// <summary>
/// Contructor of the TriggerOccurrenceInfo object
/// </summary>
/// <param name="toi_id"> TriggeOccurrenceInfo id</param>
/// <param name="toitgo_id">Id of the trigger occurrence</param>
/// <param name="toi_attribute">Trigger Occurrence Info: database atribute</param>
/// <param name="toi_value">Trigger Occurrence Info:: database value</param>
/// <param name="toi_description">Trigger Occurrence Info: description</param>
public TriggerOccurrenceInfo(int toi_id, int toitgo_id, string toi_attribute, string
     toi_value, string toi_description)
{
    id = toi_id;
    triggerOccurrenceId = toitgo_id;
    attribute = toi_attribute;
    val = toi_value;
    description = toi_description;
}

/// <summary>
/// Get/set the TriggerOccurrenceInfo id
/// </summary>
public int TriggerOccurrenceInfo_Id
{
    get
    {
        return id;
    }
    set
    {
        id = value;
    }
}

/// <summary>
/// Get/set the trigger occurrence id of the correspondent trigger occurrence
/// </summary>
public int TriggerOccurrenceInfo_TriggerOccurrenceID
{
    get
```

```
        {
            return triggerOccurrenceId;
        }
        set
        {
            triggerOccurrenceId = value;
        }
    }

    /// <summary>
    /// Get/set the TriggerOccurrenceInfo: attribute
    /// </summary>
    public string TriggerOccurrenceInfo_Attribute
    {
        get
        {
            return attribute;
        }
        set
        {
            attribute = value;
        }
    }

    /// <summary>
    /// Get/set the TriggerOccurrenceInfo: value
    /// </summary>
    public string TriggerOccurrenceInfo_Value
    {
        get
        {
            return val;
        }
        set
        {
            val = value;
        }
    }

    /// <summary>
    /// Get/set the TriggerOccurrenceInfo: description
    /// </summary>
    public string TriggerOccurrenceInfo_Description
    {
        get
        {
            return description;
        }
```

```
            set
            {
                description = value;
            }
        }
    }
```

Class TriggerParentInfo

```
public class ParentInfo
    {
        private int id;
        private int triggerOccurrenceId;
        private string attribute;
        private string val;
        private string description;

        /// <summary>
        /// Constructor of ParentInfo object
        /// </summary>
        public ParentInfo()
        {
            id = -1;
            triggerOccurrenceId = -1;
            attribute = string.Empty;
            val = string.Empty;
            description = string.Empty;
        }

        /// <summary>
        /// Contructor of the ParentInfo object
        /// </summary>
        /// <param name="tpi_id"> Parent info id</param>
        /// <param name="tpitgo_id">Id of the trigger's parent info</param>
        /// <param name="tpi_attribute">parent info: database atribute</param>
        /// <param name="tpi_value">parent info: database value</param>
        /// <param name="tpi_description">parent info: description</param>
        public ParentInfo(int tpi_id, int tpitgo_id, string tpi_attribute, string tpi_value,
               string tpi_description)
        {
            id = tpi_id;
            triggerOccurrenceId = tpitgo_id;
            attribute = tpi_attribute;
            val = tpi_value;
            description = tpi_description;
        }
```

```
/// <summary>
/// Get/set the parentinfo id
/// </summary>
public int ParentInfo_Id
{
    get
    {
        return id;
    }
    set
    {
        id = value;
    }
}

/// <summary>
/// Get/set the parentinfo's trigger id
/// </summary>
public int ParentInfo_TriggerOccurrenceID
{
    get
    {
        return triggerOccurrenceId;
    }
    set
    {
        triggerOccurrenceId = value;
    }
}

/// <summary>
/// Get/set the parentinfo: attribute
/// </summary>
public string ParentInfo_Attribute
{
    get
    {
        return attribute;
    }
    set
    {
        attribute = value;
    }
}

/// <summary>
/// Get/set the parentinfo: value
/// </summary>
```

```
    public string ParentInfo_Value
    {
        get
        {
            return val;
        }
        set
        {
            val = value;
        }
    }

    /// <summary>
    /// Get/set the parentinfo: description
    /// </summary>
    public string ParentInfo_Description
    {
        get
        {
            return description;
        }
        set
        {
            description = value;
        }
    }
}
```