

**Bob Project**

**תוכנה ליצירת XML-ים**

**מסמך תכנון ועיצוב**

**סדנה בתכנות מכוון עצמים (20586)**

**מנחה: דני כלפון**

**מגישים: אייל לוי ושאול רויזן**

אוגוסט 2014

# תיאור כללי של המערכת

המערכת בנויה ממודל 4 שכבות מופרדות זו מזו:

* שכבת ה Data Base (DB).
* שכבת ה Data Access Layer (DAL).
* שכבת ה Business Logic (BL).
* שכבת ה Graphical User Interface (GUI).

כל שכבה מתקשרת עם השכבה שמתחתיה, כאשר שכבת ה GUI היא העליונה ומכירה רק את ה BL, ה BL מעל ה DAL וה DAL מעל ה DB. בצורה זו, הממשק בין השכבות מאפשר החלפת כל אחת מהשכבות ללא שינוי בשכבות שהאחרות, בלבד שיתקיים הממשק ביניהן.

### הנחות עבודה בכתיבת הפרוייקט

TBD – אייל ושאול

### מוסכמות רישום

1. כל שם מחלקה/מאפיין/מתודה/enum שכוללת יותר ממילה אחת, נכתבת עם אות גדולה בתחילת כל מילה נוספת.
2. כל מחלקה מתחילה באות גדולה. לממשקים נוספת האות I לפני שם המחלקה.
3. כל שדה (member) פרטי במחלקה מתחיל ב-'\_m'
4. Enum-ים מתחילים באות גדולה.
5. פרמטרים למתודה או משתנים בתוך מתודה מתחילים באות קטנה.
6. אין הבדל ברישום מתודות ציבוריות (public) ומתודות פרטיות (או מוגנות) מתחילות באות גדולה. כך גם לגבי מאפיינים (Property).

# שכבת בסיס הנתונים – XML ו XSD

בשכבה זו נבנה בסיס הנתונים עם המידע לו נדרשת המערכת. זוהי שכבה שמיוצגת ע"י קבצי נתונים, בעלי מבנה מוסכם ומתוקנן, ממנו שואבים את המידע. הסבר מפורט על הרכיבים:

* **קובץ XML:** קובץ עפ"י תקנון W3C, המתאר עץ היררכי, עם צמתים ומאפיינים, המתארים טיפוסים מסוגים שונים.
* **סכמה (XSD):** קובץ XML המתאר את משפחת ה XML-ים אותה הוא מייצג, נותן חוקיות, שמות וסדר המבנה של הקבצים.

# שכבת הגישה – DAL

שכבה זו אחראית על הגישה לבסיס הנתונים, בכך שהיא קוראת את הקבצים והופכת אותם לאובייקטי C#. ישנן מחלקות המיועדות לעבודה מול XSD ומול XML, היודעות לקבל נתיב והגדרות ולייצא אובייקט C#, בהן שכבת ה DAL משתמשת. שכבת ה DAL דואגת לתת את ההגדרות המתאימות, לטפל בתקינות ולהעביר את השגיאות והאובייקטים המוכנים לשכבות מעל.

הגישה מתחלקת, כמו ה Database, לשני חלקים: גישה ל XSD וגישה ל XML-ים השונים.

מחלקות גישה ל XSD

**XsdReader.cs**

אחראית לקריאה והחזקת האובייקטים ה .NET-יים המתארים את סכמת ה XSD, אחראית על ווידוא המבנה התקין של הקובץ והעלאת שגיאות במקרה הצורך.

public XsdReader(string schemaPath)

הבנאי מקבל את נתיב קובץ ה XSD, יוצר את האובייקט ה C# עבורו ע"י מחלקות .NET שיודעות לקרוא את הנתונים ולבסוף מבצע קומפילצייה של הסכמה בתוך ה SchemaSet, על מנת לאפשר ולידציה ושימוש בנתונים. את האובייקט הסופי שומרת כמאפיין public שדרכו שכבת הלוגיקה פועלת הלאה.

public XmlSchema Schema { get; set; }

בנוסף, נשמר הנתיב של הקובץ לשימוש עתידי:

public string Path { get; set; }

פונקציית הווידוא, שקורית באתחול האובייקט, ניתנת לקריאה בכל זמן והיא סטאטית על מנת לאפשר שימוש בה לאו דווקא עם הנתיב הנוכחי, אלא על מנת לוודא סכמה שרוצים לטעון בעתיד:

public static bool ValidateSchema(string schemaPath, bool throwException)

בנוסף, קיימת פונקצייה להתאמת XML כלשהו לסכמה הטעונה במערכת. זאת על מנת לוודא שה XML שטוענים מתאים ל XSD וניתן לעבוד עימו:

public bool IsXmlMatchSchema(XmlDocument doc, out string errorMessage)

מחלקות גישה ל XML

**XmlWriterWrapper.cs**

מחלקה לכתיבת תוצר לתוך XML. זוהי מחלקה סטאטית המכילה פונקציה יחידה המקבלת את אובייקט ה XML ה .NET-י ואת הנתיב לכתיבה, ומייצאת קובץ XML על פי האובייקט הניתן לנתיב הרצוי, תוך ווידוא תקינות מיקום.

public static bool WriteXml(XmlDocument document, string destinationPath)

**XmlQueryPartType.cs**

מחלקת מבנה נתונים המתארת בקשת חיפוש של מידע בתוך XML. במחלקה המאפיינים הבאים:

public string QueriedNode { get; set; }

public string AttributeName { get; set;

public string AttributeValue { get; set; }

והם: שם הצומת שרוצים לחפש בתוכו, שם המאפיין של הצומת לעשיית השאילתה, ערך המאפיין הדרוש.

**XmlSearcher.cs**

מחלקה המטפלת בחיפוש בתוך XML אחר המידע הרצוי. בעלת פונקציה סטאטית בודדה:

public static List<XmlNode> SearchXml(XmlDocument document, XmlQueryPartType query)

מחפשת בתוך XML את הנתונים הנדרשים עפ"י השאילתה המבוקשת, ומחזירה רשימה של צמתים העונים על הדרישה.

**XmlLoaderWrapper.cs**

מחלקה לטעינת XML מתוך קובץ לתוך אובייקט .NET סטנדרטי. בעלת פונקציה סטאטית בודדה:

public static XmlDocument LoadXml(string xmlPath)

יוצרת ומחזירה את אובייקט ה .NET המתאים תוך ווידוא שגיאות.

# שכבת הלוגיקה – BL

שכבה זו מקשרת בין שכבת קריאת בסיס הנתונים, ה DAL, לבין שכבת התצוגה, ה GUI. תפקידה העיקרי של השכבה הוא לעטוף את האובייקטים ה .NET-יים שה DAL מספק לה ולאפשר גישה נוחה לנתונים, מניפולציה שלהם, ניתוח המידע וחשיפת ממשק לשכבת התצוגה הפשוט ביותר.

***עקרונות מנחים***

* ה BL תייצא את המידע הדרוש בלבד מתוך אובייקטי ה DAL.
* כלל ה Exceptions, הן מה DAL והן מה BL עצמו, יועלו הלאה את שכבת ה GUI על מנת לאפשר הצגת שגיאות למשתמש.
* עיקרון הכימוס (Encapsulation) בא לידי ביטוי בשכבה זו, בכך שהיא מסתיר את המידע הנשמר במחלקות וחושפת אותו כלפי חוץ באמצעות תכונות (Properties) בלבד. הדבר נעשה גם אם אין הגבלות מסוימות על הערך של המידע ולא מתבצעת שום בדיקה בגוף של התכונה, וזאת כדי לאפשר בעתיד הוספה של בדיקות כאלו לכל מידע שנשמר, בלי צורך לשכתב קוד קיים במקומות אחרים.
* עקרון ה SRP בא לידי ביטוי ע"י כך שהאחריות בין המחלקות השונות, כאשר כל מחלקה מקבלת תפקיד ספציפי ורק הממשק כלפי ה GUI מכיל את כל שאר יכולות ה BL.
* עקרון הפולימורפיזם בא לידי ביטוי בכך שישנן מחלקות abstract מהן יורשות מספר מחלקות ומאפשרות להשתמש בהן מבלי לדעת את המימוש הספציפי של כל אחת מהן או אפילו מאיזה טיפוס מדויק הן.
* מחלקות מכילות מחלקות אחרות על מנת להימנע מירושה מסורבלת ולהרחיב יכולות, תוך שימוש ב Design Patterns שונים.

מחלקות כלליות

**ConfigurationData.cs**

TBD - אייל

**DateRange.cs**

זוהי מחלקה לטיפול בטווחי תאריכים. השימוש בה נועד עבור ייצור דו"חות, שבהם ניתן לבחור דו"ח עם טווח תאריכים ולקבל פלט מתאים.

במחלקה שני מאפיינים לסימון תאריך התחלה ותאריך סיום בשאילתה המבוקשת:

public DateTime Start

public DateTime End

בנוסף, ישנן פונקציות לעבודה עם התאריכים.

הדפסת מחרוזת יפה לעין של טווח התאריכים (העמסה מה ToString של האובייקט הבסיסי):

public override string ToString()

הפיכה לאובייקט TimeSpan לעבודה של חישובים:

public TimeSpan ToTimeSpan()

בדיקה האם התאריך או התאריכים הנתון נמצאים בטווח הרצוי:

public bool IsInRange(DateTime date)

public bool IsInRange(params DateTime[] dates)

public bool IsInRange(DateRange dates)

**ObservableDictionary.cs**

TBD - אייל

**Permission.cs**

TBD - אייל

**INotifyHighLevelPropertyChanged.cs**

ממשק המאפשר עדכון לגבי שינויים במחלקות הנתונים של מאפיינים שאינם פשוטים, כלומר המכילים מאפיינים שתלויים במאפיינים אחרים וצריכים להתעדכן בכל פעם שאחד ממאפיינים אלו מתעדכן. הממשק מכיל אירוע אחד:

event PropertyChangedEventHandler HighLevelPropertyChanged

אירוע זה מוקפץ ע"י המחלקה הממשמשת את הממשק בכל שינוי של מאפיין פשוט שבו תלוי מאפיין אחר, ובכך הנרשמים יודעים על השינויים של כל המאפיינים הרלוונטיים.

**PropertyNotifyObject.cs**

TBD - אייל

**XmlMetaData.cs**

זוהי מחלקת מידע, המכילה תיאור של מידע צד על ה XML. המאפיינים הינם נתיב קובץ ה XML, גרסת ה XML, תאריך עדכון אחרון ושם המשתמש שיצר את ה XML.

public string XmlPath

public Version Version

public DateTime Date

public string UserName

מחלקות לעטיפת אובייקטי Schema

**XmlSchemaWrapper.cs**

מחלקה אבסטרקטית מרכזית המתארת את האובייקט הכללי של סכמת XSD, עם המאפיינים והפעולות הדרושים עבור Bob. האבסטרקציה הזו מאפשרת למחלקות המכילות אובייקטים מסוג היורשים ממנה, ללא צורך לדעת מהו המימוש המסוים. דבר זה חשוב עבור המעבר על עץ הטיפוסים. אובייקט כללי של XSD הוא תיאור של צומת בקובץ ה XSD, עם כל המידע הדרוש לעבודה עימו הניתן ע"י שכבת ה DAL. האובייקט בנוי בצורת עץ, כלומר כל צומת מכילה רשימת ילדים ואת האב שלה, כאשר לשורש אין אב, ולעלה אין ילדים.

בנוסף, העיצוב מתוכנן כך, שכלל הטיפוסים מסוג XmlSchemaWrapper ישאירו מקום למילוי המידע ע"י המשתמש ושמירתו בתוכם. פירוט בכל טיפוס יורש.

*מאפיינים:*

שם הצומת, סוג הצומת, רשימת ילדים, אב הינם המידע הבסיסי לזיהוי הצומת:

public string Name

public NodeType NodeType

public ObservableCollection<XmlSchemaWrapper> Children

public XmlSchemaWrapper Parent

מבנה העץ של סכמה יכול להיות מאוד ארוך ומסועף, וכן ישנה דרישה לתמוך בעץ רקורסיבי, כלומר טיפוס כלשהו יכול להכיל צאצא שהוא מאותו טיפוס שלו. מכאן שאין אפשרות לקרוא את כל עץ הסכמה במעבר יחיד, אלא יש לחכות לבקשות של היכנסות פנימה לתוך הטיפוס הנבחר מתוך אפשרויות הבחירה. הסיבה לכך היא יכולת ה choice שבתוך ה XSD, וכך ניתן לבחור טיפוס פשוט או מסועף בעל תת-עץ ארוך בפני עצמו.

מסיבה זו, ישנו שימוש נרחב בתוכנה במנגנון ה Lazy Loading, שאומר שטוענים רמה אחת בכל פעם, מחכים לדרישה (שמגיעה מהמשתמש) ולפיה ממשיכים לקרוא את המידע הדרוש עד שמגיעים לתנאי עצירה והוא טיפוס פשוט.

לכן ישנו חיווי בכל אחד מהאובייקטים לגבי האם הוא ניתן להמשך "חפירה", Drill ובנוסף, במידה והוא ניתן לחפירה, האם הוא כבר "נחפר". אלה הם המאפיינים:

public abstract bool IsDrillable

public bool HasBeenDrilled

כל תת-טיפוס היורש ממחלקה זו קובע כיצד יודעים אם האובייקט הוא ניתן לחפירה.

בנוסף, קיימים שני מאפיינים לבדיקת תקינות כל ילדיו בצורה רקורסיבית. כלומר, המאפיין מציין האם התנאי מתקיים על כל אחד מהצמתים בתת העץ שלו. מאפיין אחד הוא האם כל ילדיו "נחפרו", והשני האם כל ה attributes ההכרחיים של תת העץ שלו (הרלוונטיים רק עבור elements, הסבר בהמשך), מלאים ותקינים.

public virtual bool AllChildrenDrilled

public virtual bool AllChildAttributesFilled

*בנאי*

אתחול האובייקט נעשה עם הבנאי הבא:

public XmlSchemaWrapper(string name, NodeType nodeType, XmlSchemaWrapper parent, bool nonDrillable = false)

מקבל את שם הצומת, כפי שמופיע ב XSD, למשל element name="aggr".

טיפוסה הוא מאחד הסוגים שבהם מערכת Bob תומכת מתוך הטיפוסים של ה XSD. הטיפוסים מתוארים באמצעות enum:

public enum NodeType { Element, Choice, Sequence, SequenceItem, NULL }

שמתוכו ניתן לדעת עם איזה סוג טיפוס אנו עובדים, אפילו כאשר עובדים עם האובייקט האבסטרקטי.

הפרמטר הבא הוא אב הצומת, שמכיל את הצומת בתוך מערך ה Children שלו.

הפרמטר האחרון הוא אופציונאלי ובברירת מחדל הוא "שקר", המציין האם הוא לא ניתן לחפירה, דבר המשפיע על המאפיין HasBeenDrilled, שכן אובייקט ללא אפשרות חפירה יעדכן את הדגל ל"אמת".

*פונקציות*

פונקציה לעדכון כלל המאפיינים שהם High Level, כלומר המתעדכנים בהתאם למאפיינים אחרים:

protected virtual void RaiseAllProperties()

פונקציה אבסטרקטית לביצוע "חפירה", כך שכל אובייקט יורש קובע כיצד זה ייעשה:

protected abstract void InternalDrill()

פונקציה לביצוע החפירה עצמה בעומק אחד, ממשמשת תבנית עיצוב של Decorator, בכך שיש לה פעולות קבועות שהיא מבצעת וקוראת בזמן הנכון ל InternalDrill אותו ממשמשות המחלקות היורשות:

public void DrillOnce()

לבסוף, פונקציה לייצוג יפה של המחרוזת של האובייקט, להדפסת השם שהתקבל בבנאי:

public override string ToString()

**XmlSchemaElementWrapper.cs**

מחלקה המתארת ומבצעת את הפונקציונאליות של Element בתוך הסכמה. ה Element הינו צומת מרכזי ב XSD, לו יש תכונות, Attributes. בנוסף, הוא יכול להיות מטיפוס פשוט, כלומר עלה בעץ, או מטיפוס מורכב, צומת עם תת עץ משלו. ה Element יורש מה XmlSchemaWrapper ומוסיף עליו את תכונותיו ופעולותיו המיוחדות.

*מאפיינים*

האובייקט ה .NET-י אותו עוטפים, ממנו שואבים את המידע אודות הצומת:

private XmlSchemaElement ElementObject

הטיפוס (פשוט או מורכב) עם כלל המידע האצור בתוך אובייקט הטיפוס (יפורט בהמשך):

private IXmlSchemaTypeWrapper Type

דגל המורה האם האיפוס הוא מסוג פשוט (כדי למנוע מהמשתמשים לבדוק את המאפיין Type בכל פעם):

public bool IsSimple

אם זהו טיפוס פשוט, יש לו תרגום .NET-י, למשל int או string שבו ממלאים אותו.

public Type DotNetType { get; private set; }

מאפיין חשוב נוסף ב XSD עבור ה Element הוא מספר המופעים שלו, המינימאלי והמקסימאלי. משתמשים במאפיין כאשר הוא נמצא בתוך טיפוס מסוג מערך.

public decimal MinOccurs { get; set; }

public string MaxOccursString { get; set; }

דגל מיוחד שיש ל XmlSchemaElementWrapper הוא האם כל התכונות (attributes) הכרחיות שלו עצמו מלאות:

public bool AllAttributesFilled

שני דגלים שמקבלים מהירושה, ודורסים אותם על מנת לבצע את המימוש המתאים ל Element, והם האם כל הילדים (כולל עצמו) מילאו את התכונות (attributes) ההכרחיות וכן האם הטיפוס הוא ניתן ל"חפירה", כלומר יש לו תת עץ משלו:

public override bool AllChildAttributesFilled

public override bool IsDrillable

לבסוף, עבור ה Element יש כמה מאפיינים שממלאים אותם על ידי המשתמש. האחד הוא הטקסט הפנימי שיימצא בתוך צומת ה XML, בין ה TAGS של הצומת. הדבר רלוונטי רק עבור טיפוס פשוט, לו אין תת עץ אלא תת עץ פנימי:

public string InnerText

המאפיין השני הוא רשימת התכונות, ה Attributes. זוהי רשימה שנקראת באתחול מהסכמה ונשאר מקום למילוי הערכים שלהן:

public ObservableCollection<XmlSchemaAttributeInfo> Attributes

*בנאי*

public XmlSchemaElementWrapper(XmlSchemaElement element, XmlSchemaWrapper parent) :

base(element.Name, NodeType.Element, parent)

הבנאי מקבל את האובייקט ה .NET-י של ה Element, ממנו שואבים את המידע ואותו שומרים, וכן את האב של ה Element כפי שיש עבור כל אובייקט מטיפוס XmlSchemaWrapper. אל המחלקה הבסיסית שלו הוא מעביר את שם ה Element, מציין שהוא מסוג Element ע"י ה enum וכן את האב שקיבל. במהלך הבנאי הוא מאתחל את המאפיינים השונים על פי האובייקט ה .NET-י.

*פונקציות*

על האובייקט לממש את הפונקציות האבסטרקטיות שאביו מייצא. האחת היא העלאת הודעה על שינוי מאפיינים High Level, כלומר אלו התלויים באחרים. המאפיין הייחודי במחלקה זו הינו AllAttributesFilled. השנייה היא פונקצית ה"חפירה" אל תוך הטיפוס, אותה יבצע ע"י קריאה מתוך מחלקת האב. החפירה תתבצע רק אם הטיפוס הוא מסוג מורכב.

protected override void RaiseAllProperties() (

protected override void InternalDrill()

**XmlSchemaGroupBaseWrapper.cs**

מחלקה אבסטרקטית לתיאור טיפוס סכמה המייצג קבוצה. בתוך שפת ה XSD ישנם מספר טיפוסי קבוצה, הבולטים מביניהם הם ה Sequence וה Choice. אלו הם טיפוסים המכילים בתוכם תתי טיפוסים.

*מאפיינים*

דריסת מאפיין המציין האם ניתן לחפור בתוך האובייקט. תמיד יחזיר "אמת", מכיוון שזוהי הגדרתו של טיפוס קבוצה, יש לו בוודאות תת-עץ.

public override bool IsDrillable

שמירת הטיפוס ה .NET-י ממנו שואבים את המידע של הקבוצה:

private XmlSchemaGroupBase Group

*בנאי*

מקבל את הטיפוס ה .NET-י, סוג הטיפוס הספציפי (מתקבל מהטיפוס היורש) את אביו של הצומת. מאתחל את השם, הטיפוס ואב של ה XmlSchemaWrapper.

public XmlSchemaGroupBaseWrapper(XmlSchemaGroupBase group, NodeType nodeType, XmlSchemaWrapper parent)

: base(nodeType.ToString(), nodeType, parent)

*פונקציות*

דריסת ה"חפירה" הספציפית עבור טיפוס קבוצה. במקרה של קבוצה, יש למלא את הילדים של הצומת ועבור כל ילד לבדוק מאיזה סוג הוא ולאתחל אותו בהתאם.

protected override void InternalDrill()

פונקציה סטאטית, הממשמשת תבנית עיצוב של Factory Method, המייצרת מתוך טיפוס קבוצה כללי (הטיפוס של המחלקה XmlSchemaGroupBaseWrapper) את הטיפוס הרצוי על פי הטיפוס ה .NET-י שהועבר.

public static XmlSchemaGroupBaseWrapper SchemaGroupWrappersFactory(XmlSchemaGroupBase group, XmlSchemaWrapper parent)

*אירועים*

בעת שמבצעים את ה Drill בקבוצה, ייתכן שיהיו לקוחות שירצו לדעת על כך ולבצע פעולות בהתאם. האירוע מוקפץ כאשר עושים "חפירה"

protected event Action OnGroupDrill

**XmlSchemaSequenceArray.cs**

בתוך סכמת XSD, קיים טיפוס מרכזי בשם Sequence. זהו מערך של טיפוסים, כלומר כזה שניתן ליצור מספר כלשהו של מופעים עבור קבוצת הטיפוסים שתחתיו. בכך הוא שונה מטיפוסים אחרים ב XSD, לא רק שיש לו תת-עץ, אלא זהו תת-עץ עם מספר מופעים, מערך. טיפוס כל אחד מהמופעים הוא זהה (יפורט ב XmlSchemaSequenceWrapper). על מנת לממש זאת, המחלקה צריכה גם להוות טיפוס XSD כמו שאר המחלקות מסוגה וגם להוות מערך .NET-י נוח לשימוש. לכן הוא יורש מטיפוס קבוצה וכן מממש את הממשק הסטנדרטי של מערך, מסוג טיפוסי הסכמה:

public class XmlSchemaSequenceArray : XmlSchemaGroupBaseWrapper, IList<XmlSchemaSequenceWrapper>

*בנאי*

מאותחל בטיפוס הסטנדרטי של כל אחד מאיבריו וכן נשלח אליו האב שלו כמו שאר הטיפוסים של XSD. מאתחל את הטיפוס הבסיסי באמצעות סוג איברי המערך, ציון סוגו עצמו (ע"י ה enum) ושליחת אביו:

public XmlSchemaSequenceArray(XmlSchemaSequence sequence, XmlSchemaWrapper parent)

: base(sequence, NodeType.Sequence, parent)

*מימוש ממשק מערך*

זהו ממשק סטנדרטי של IList עם פעולות ומאפייני המערך הרגילים:

public int IndexOf(XmlSchemaSequenceWrapper item)

public void Insert(int index, XmlSchemaSequenceWrapper item)

public void RemoveAt(int index)

public XmlSchemaSequenceWrapper this[int index]

public void Add(XmlSchemaSequenceWrapper item)

public void Clear()

public bool Contains(XmlSchemaSequenceWrapper item)

public void CopyTo(XmlSchemaSequenceWrapper[] array, int arrayIndex)

public int Count

public bool IsReadOnly

public bool Remove(XmlSchemaSequenceWrapper item)

public IEnumerator<XmlSchemaSequenceWrapper> GetEnumerator()

*פונקציות*

מימוש ה"חפירה" בתוך האובייקט, שצריכה לעשות זאת עבור כל אחד מאיברי המערך:

protected override void InternalDrill()

פונקציה להוספת איבר נוסף במערך, מהטיפוס הידוע שלו, מסוג XmlSchemaSequenceWrapper:

public void AddNewWrapper()

**XmlSchemaSequenceWrapper.cs**

זהו הטיפוס המרכיב את איברי המערך של XmlSchemaSequenceArray. יורש גם כן מ GroupBaseWrapper מכיוון שמכיל בתוכו תת-עץ טיפוסים.

*בנאי*

מקבל, בנוסף לפרמטרים הנדרשים עבור טיפוס הבסיס, את האינדקס, מיקומו במערך המכיל אותו, זאת בשביל תצוגה נוחה למשתמש בעץ הטיפוסים.

public XmlSchemaSequenceWrapper(XmlSchemaSequence sequence, XmlSchemaSequenceArray parent, int index)

: base(sequence, NodeType.SequenceItem, parent)

*מאפיינים*

אינדקס במערך הקבוצה, שומרים אותו בשביל תצוגה נוחה למשתמש:

public int Index

*פונקציות*

דריסת פונקצית התצוגה, כך שיילקח מיקומו במערך:

public override string ToString()

**XmlSchemaChoiceWrapper.cs**

מחלקה לתיאור טיפוס מסוג Choice בתוך סכמת XSD. טיפוס זה דומה ל Sequence ולקבוצות באופן כללי בכך שיש לו תת-עץ טיפוסים. עם זאת, תת העץ הזה מהווה רשימת טיפוסים מתוכן ניתן לבחור את הטיפוס הדרוש למשתמש. כלומר, ההבדל מ Sequence שכאן נבחר טיפוס אחד מתוך הרשימה ולא כולם ביחד, וכן זהו מופע בודד שלו. לכן הוא גם כן יורש מהטיפוס XmlSchemaGroupBaseWrapper, כפי שזה בא לידי ביטוי באובייקט ה .NET-י.

*מאפיינים*

שני מאפיינים שמתקבלים בירושה הם: האם כלל התכונות (attributes) בתת העץ שלו מלאים, והשני האם בכל הילדים שלו נעשתה "חפירה". במקרה של Choice עלינו לוודא זאת רק עבור הטיפוס הנבחר ולא על כל הילדים האפשריים שלו.

public override bool AllChildAttributesFilled

public override bool AllChildrenDrilled

בנוסף, עלינו לשמור ולתחזק את הטיפוס הנבחר, ע"י המשתמש. כאשר ישנו שינוי בטיפוס הנבחר, הדבר משפיע על המאפיינים של הטיפוס כולו.

public XmlSchemaWrapper Selected

*בנאי*

הבנאי הוא סטנדרטי עבור טיפוס קבוצה, מקבל את האובייקט ה .NET-י המייצג אותו ואת אביו ושולח הלאה אל מחלקת האב.

public XmlSchemaChoiceWrapper(XmlSchemaChoice choice, XmlSchemaWrapper parent)

: base(choice, NodeType.Choice, parent)

*פונקציות*

המתודה הראשונה היא לתצוגה למשתמש, שם אנו מציגים שזהו Choice עם הטיפוס הנבחר:

public override string ToString()

השנייה היא לביצוע ה"חפירה" כפי שצריך עבור ה Choice:

protected override void InternalDrill()

**XmlSchemaNullChoice.cs**

זוהי מחלקה המהווה ברירת מחדל עבור Choice. ה Choice הוא בעל תכונה האומרת האם הוא יכול להיות NULL, כלומר המשתמש יכול שלא להשתמש ב Choice כלל ולהשאירו ריק. אם כן, עלינו למלא את ה Choice על מנת לשמור על המבנה הסטנדרטי של העץ, וזהו הטיפוס לכך. קביעתו כנבחר עבור ה Choice אומר שאין צורך בכתיבתו ב XML הנוצר.

*בנאי*

מאתחל את מחלקת האב (XmlSchemaWrapper) עם שם "NULL", טיפוס זה גם ב enum, אביו (שהוא בהכרח Choice) וציון שזהו טיפוס שלא ניתן לעשות לו "חפירה".

public XmlSchemaNullChoice(XmlSchemaChoiceWrapper parent)

: base("NULL", NodeType.NULL, parent, true)

*מאפיינים*

מאפיין האומר האם ניתן לעשות "חפירה". תמיד יהיה "שקר":

public override bool IsDrillable

*פונקציות*

מימוש הפונקציה האבסטרקטית של "חפירה". צריכה להיות ריקה:

protected override void InternalDrill()

**XmlSchemaAttributeInfo.cs**

מחלקה לתיאור תכונה (Attribute) של Element.

*מאפיינים*

שם התכונה:

public string Name

שימוש בתכונה (הכרחית, אופציונאלית, ללא שימוש):

public XmlSchemaUse Use

האם נדרשת, כלומר הכרחית עבור אותו Element:

public bool IsRequired

האם מולאה כמו שצריך?

public bool IsAttributeValid

האם תקינה, במקרה זה יכול להיות שהיא לא מולאה בכלל אך מכיוון שהיא לא הכרחית, תחזיר "אמת":

public bool IsAttributeFilled

הטיפוס ה .NET-י שלה:

public Type SimpleType

הערך שבו מולאה. הערך הוא תמיד מחרוזת:

public string Value

*בנאי*

יצירת תכונה חדשה:

public XmlSchemaAttributeInfo()

*אירועים*

אירוע המציין שקרה שינוי במאפיין שהוא High Level, כלומר מאפיין בעל תלות במאפיינים אחרים. משמש ליידוע השכבות השונות על שינוי הערך ותגובה בהתאם.

public event PropertyChangedEventHandler HighLevelPropertyChanged

מחלקות לעטיפת טיפוסי אלמנטים בסכמה

המחלקות לעטיפת טיפוסי Elements יכולות להיות מסוג פשוט ומסוג מורכב. המבנה עם הממשק מאפשר יצירתן באמצעות Factory Method.

**IXmlSchemaTypeWrapper.cs**

ממשק שנועד לתיאור של טיפוס הכללי, מאפיינים המשותפים לשני הסוגים.

המאפיין היחיד הוא שם הטיפוס:

string Name { get; }

**XmlSchemaSimpleTypeWrapper.cs**

מתאר טיפוס Element פשוט.

*מאפיינים*

המאפיינים הם: שם הטיפוס, האובייקט ה .NET-י שאותו עוטפים, הטיפוס ה .NET-י שמייצג הטיפוס הפשוט ותבנית המילוי של הטיפוס, לביצוע ולידציה:

public string Name { get; private set; }

public XmlSchemaSimpleType SchemaType { get; private set; }

public Type DotNetType { get; private set; }

public string Pattern { get; private set; }

*בנאי*

מקבל רק את הטיפוס ה .NET-י על מנת לשאוב ממנו את כל שאר המידע:

public XmlSchemaSimpleTypeWrapper(XmlSchemaSimpleType type)

*פונקציות*

ה Factory Method, פונקציה סטאטית, שיוצרת את הטיפוס הרלוונטי על פי הדרישה, כך שה Element לא צריך לדעת מאיזה טיפוס מדויק הוא:

public static IXmlSchemaTypeWrapper SchemaWrappersFactory(XmlSchemaType baseType)

פונקציה סטאטית לחילוץ הטיפוס ה .NET-י מתוך טיפוס ה Element שניתן לו:

public static Type GetDotNetType(IXmlSchemaTypeWrapper type)

**XmlSchemaComplexTypeWrapper.cs**

מתאר טיפוס Element מורכב, כזה שיש לו תת-עץ מתחתיו.

*מאפיינים*

שם הטיפוס:

public string Name

האובייקט ה .NET-י לתיאור הטיפוס ממנו שואבים את המידע:

public XmlSchemaComplexType SchemaType

הטיפוס הפנימי של הטיפוס המורכב. כאן באה לידי ביטוי תבנית העיצוב Composite, מכיוון שהטיפוס הפנימי הוא מסוג טיפוס בעצמו, וכך נבנה העץ עם צמתים (טיפוס מורכב( ועלים (טיפוס פשוט):

public IXmlSchemaTypeWrapper InnerType

*בנאי*

מקבל את הטיפוס ה .NET לשאיבת המידע הדרוש:

public XmlSchemaComplexTypeWrapper(XmlSchemaComplexType complexType)

*פונקציות*

זוהי פונקציה היחידה של הטיפוס, סטאטית והיא אחראית לשאיבת התכונות (attributes) של הטיפוס. אם זהו טיפוס מורכב, יש לו תכונות ואף עשויות להיות בטיפוס הפנימי שלו, ממנו הוא יורש, לכן יש לעשות זאת באופן רקורסיבי עד מציאת כלל התכונות של הטיפוס:

public static ObservableCollection<XmlSchemaAttributeInfo> GetAllAttributes(IXmlSchemaTypeWrapper type)

מחלקת הממשק הראשית

**SchemaDescriber.cs**

זוהי מחלקת הממשק הראשי, זו שדרכה מבוצעות כלל הפעולות בין ה BL לשכבות שמעליה ומתחתיה. המחלקה מכילה מאפיינים ופונקציות שחושפות את הפונקציונאליות של המערכת וכל משתמש בממשק יוכל להגיע דרכה ליכולות הלוגיות של המערכת. באותו הקשר, יש לה שתי יכולות ראשיות שהיא מכילה: עבודה מול הסכמה, ה XSD, וכן עבודה וייצור של XML מתוך הסכמה הזו.

*מאפיינים*

רשימת ה Elements המהווים את השורש של ה XSD. לרוב יכיל רק Element אחד.

public ObservableCollection<XmlSchemaElementWrapper> Elements

אלמנט השורש של ה XSD:

public XmlSchemaElementWrapper RootElement

האובייקט של ה XML שכרגע בעריכה:

public XmlDocument CurrentXmlDocument

גרסה, שם משתמש ותאריך עדכון של ה XML כשרגע בעריכה:

public Version XmlVersion

public string UserName

public DateTime LastEditDate

*בנאי*

מקבל נתיב לסכמה ואת שם המשתמש הנוכחי ואחראי לטעון את הסכמה:

public SchemaDescriber(string schemaPath, string userName = null)

*פונקציות*

טעינת סכמה מנתיב נתון, ווידוא תקינותה ויצירת המאפיינים השונים:

public void LoadSchema(string schemaPath)

ווידוא תקינות הסכמה. ניתן להשתמש בה עם זריקת חריגה או בלי:

public bool ValidateSchema(string schemaPath, bool throwException = false)

פונקציה להחזרת אובייקט ה XML הנוכחי שבעריכה:

public XmlDocument GetCurrentXmlDocument()

מחיקת כלל הנתונים שהוכנסו עד כה לתוך האובייקטים השונים (תכונות, טקסט פנימי, אובייקטים נבחרים, איברים במערך וכו'):

public void ClearXml()

ייצוא ה XML במצבו הנוכחי. מקבל את הגרסה ושם המשתמש בנוסף לנתיב הרצוי של ה XML:

public bool ExportXmlNow(string xmlPath, Version version = null, string userName = null)

חיפוש בתוך ה XML עם ההגדרות שמשתמש בחר, שהם הצומת ממנו מתחילים לחפש, מחרוזת השאילתה וצורת החיפוש (בתוך תכונות או שמות צמתים):

public List<XmlSchemaWrapper> SearchXml(XmlSchemaWrapper startingNode, string query, SearchEnum searchBy)

יצירת דו"חות על פי המבוקש. הפונקציה מחפשת בתוך התיקייה הנתונה קבצי XML התואמים לסכמה, ומחפשת את המשתמש והתאריכים הרצויים:

public ObservableCollection<XmlMetaData> ProduceReport(string folderPath, string userName, DateRange dates)

מחלקות לעטיפת יכולות XML

**XmlExportLogic.cs**

TBD - שאול

**XmlWrappersSearcher.cs**

TBD - שאול