

**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 הפרויקט

1. הגדרות המערכת נשמרות בRegistry. ההגדרות כוללות: הרשאות, נתיב הסכמה ומקרא הצבעים.
2. ישנם 3 סוגים שונים של הרשאות במערכת והן: מנהל מערכת, עורך וצופה.
3. המערכת אינה תומכת ברישום משתמשים חדשים למערכת.
4. מנהל מערכת הוא בעל ההרשאה החזקה ביותר ויכול לבצע כל פעולה במערכת. העורך יכול לקרוא ולייצור קבצי XML אך אינו יכול לשנות הגדרות חדשות בRegistry ואינו יכול לצפות בדוחות מערכת. הצופה יכול רק לצפות בקבצי XML קיימים.
5. העץ של סכמה יכול להיות מאוד ארוך ומסועף ולכן טעינת העץ היא לפי דרישה (Lazy load).

### 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).

במסמך זה נגדיר את השמות הבאים:

* Exception ייקרא "חריגה".
* Constructor ייקרא "בנאי".
* Interface ייקרא "ממשק".
* Property של מחלקה ייקרא "מאפיין".

# 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.

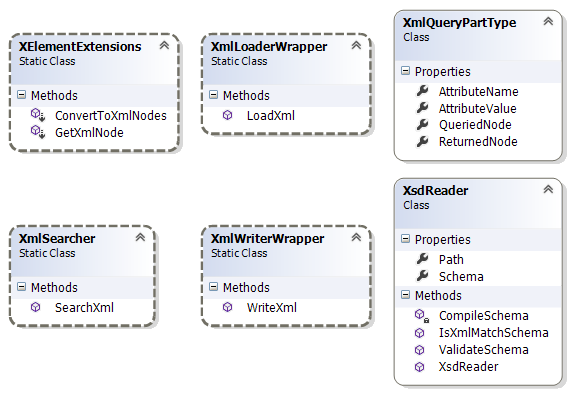
**XmlLoaderWrapper.cs**

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.

דיאגרמת מחלקות עבור DAL



# 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**

מחלקה סינגלטונית (singleton) לקריאת הגדרות מערכת מהRegistry. השימוש בה נועד עבור קריאת רשימת הצבעים של סוגי הטיפוסים ששכבת ההצגה תצבע אותם וקריאת נתיב הסכמה.

*Enum and consts*

המחלקה מכילה שדות פרטיים המתארים את הנתיב בתוך הRegistry ובנוסף enum המתאר מאיזה שדה לקרוא/לכתוב

public enum Regkeys { Choice, ChoiceNull, Element, Error, Sequence, SchemaPath, Search };

private const string pathColorConf = "Bob\\Colors";

private const string pathSchemaConf = "Bob\\Configuration";

*מאפיינים*

בנוסף, המחלקה מכילה את המאפיינים השומרים את רשימת הצבעים, נתיב הסכמה, האם חלה שגיאה במהלך קריאת הנתונים והמופע של המחלקה (המחלקה היא singleton).

public ObservableDictionary<string, Color> TypesColor

public String SchemaPath

public bool IsErrorLoadingColors

public bool IsErrorLoadingSchema

public static ConfigurationData Instance

*פונקציות*

בנוסף, ישנן פונקציות פרטיות הממלאות את מאפייני המחלקה, כלומר קוראת את ההגדרות מהregistry בהתאם לEnum.

private void ReadConfigurationFromRegistry()

private string GetSetRegisterValue(Regkeys key, string value = null, bool isGet = true)

בנוסף, ישנו פונקציה השומרת את הנתונים החדשים בregistry. בהתאם למה שנשמר במאפייני המחלקה.

public void SaveConfig()

**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**

מחלקה זו הינה שילוב של המחלקות ObservableCollection וDictionary.

כלומר, המחלקה הינה רשימה בה כל איבר ברשימה הוא Dictionary עם מפתח וערך.

לכן, הגדרת המחלקה היא:

public class ObservableDictionary<TKey, TValue> :

IDictionary<TKey, TValue>,

ICollection<KeyValuePair<TKey, TValue>>,

IEnumerable<KeyValuePair<TKey, TValue>>,

IDictionary,

ICollection,

IEnumerable,

ISerializable,

IDeserializationCallback,

INotifyCollectionChanged,

INotifyPropertyChanged

המחלקה מכילה את אותם פונקציות של ObservableCollection.

**Permission.cs**

מחלקה סינגלטונית (singleton) לקריאת משתמשי המערכת מהRegistry. ישנם 3 משתמשים במערכת: מנהל מערכת, עורך וצופה.

*Enum and consts*

המחלקה מכילה שדה פרטי המתאר את הנתיב בתוך הRegistry ובנוסף שני enum: אחד המתאר את סוגי המשתמשים השונים והשני מתאר את שם השדות בregistry

public enum PermissionType { Viewer, Editor, Manager };

private enum Regkeys { EditorUser, ViewerUser, ManagerUser, EditorPass, ViewerPass, ManagerPass };

private const string pathReg = "Bob\\Users";

*מאפיינים*

בנוסף, המחלקה מכילה את המאפיינים השומרים את ההרשאה הנוכחית של המערכת, האם חלה שגיאה במהלך קריאת הנתונים והמופע של המחלקה (המחלקה היא singleton).

public bool IsErrorLoading

public PermissionType CurrPermission

public static Permission Instance

*פונקציות*

בנוסף, ישנן פונקציות פרטיות הממלאות את מאפייני המחלקה, כלומר קוראת את ההגדרות מהregistry בהתאם לEnum.

private void ReadPermissionFromRegistry()

private string GetRegisterValue(Regkeys key)

בנוסף, ישנו פונקציה המקבלת שם משתמש וסיסמא ובודקת האם זה תואם לאחד ההרשאות שבמערכת. במידה וכן, תעדכן את מאפיין CurrPermissionותחזיר אמת. אחרת, שקר.

public bool CheckPermission(string user, string pass)

**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**

מחלקה שתפקידה לעלות אירוע כאשר המאפיין השתנה. ובכך הערך ב-GUI ישתנה באמצעות מנגנון ה-binding.

מחלקה זו יורשת ממשק INotifyPropertyChangedולכן ממששת את הפונקציות שלה.

*אירוע*

public event PropertyChangedEventHandler PropertyChanged;

*פונקציה*

protected void RaisePropertyChangedEvent(string propertyName)

מטודה המקבלת את שם המאפיין שהשתנה (propertyName) ומעלה את האירוע PropertyChanged על שם המאפיין שהשתנה.

protected void SetProperty<T>(ref T field, T value, [CallerMemberName] string name = "")

מטודה תבניתית אשר מקבלת אובייקט וערך חדש ומציבה את הערך באובייקט. הפונקציה מעלה את האירוע PropertyChangedEventHandler במידה והערך השתנה.

**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**

מחלקה סטאטית האחראית על ייצוא ה XML לאחר שהוא מוכן. מכיוון שזוהי מחלקת שכבת BL, ההתעסקות היא עם אובייקטי ה Wrappers.

public static XmlDocument SchemaWrapperToXmlDocument(XmlSchemaElementWrapper rootWrapper, Version version, string userName = null, bool validate = true)

הפונקציה שמבצעת את תרגום האובייקט ה BL-י לאובייקט ה DAL-י. מקבלת את השורש ממנו מתחילים את המעבר על העץ, את נתוני הרקע על הקובץ (גרסה ושם המשתמש) ואפשרות האם לוודא את תקינות כל האובייקטים. האפשרות של לייצר אובייקט ללא ווידוא נועדה על מנת להציג תוצר חלקי בזמן העריכה.

**XmlWrappersSearcher.cs**

מחלקה האחראית על חיפוש בתוך XML. החיפושים הנדרשים במערכם הינם על פי שם של צומת כלשהו או על פי ערך של תכונה כלשהי.

*מאפיינים*

הפונקציה להשוואת מחרוזות שבהן תשתמש המחלקה לקבלת התאמה. ברירת המחדל היא פונקציה שבודקת האם המחרוזת הראשונה מכילה את המחרוזת השנייה.

public Func<string, string, bool> StringsCompareFunc

*בנאי*

מקבל פרמטר יחיד, אופציונאלי, שהוא פונקציה להשוואת המחרוזות. אם לא התקבלה פונקציה כפרמטר – פונקציית ברירת המחדל תהיה בשימוש.

public XmlWrappersSearcher(Func<string, string, bool> compareFunc = null)

*פונקציות*

חיפוש על פי תכונה. מקבלת את הצומת ממנו מתחילים לחפש ואת המחרוזת לחיפוש. מחזירים רשימה של אובייקטים תואמים:

public List<XmlSchemaWrapper> SearchXmlByAttribute(XmlSchemaWrapper startingNode, string attributeValue)

חיפוש על פי שם צומת. מקבלת את הצומת ממנו מתחילים לחפש ואת המחרוזת לחיפוש. מחזירים רשימה של אובייקטים תואמים:

public List<XmlSchemaWrapper> SearchXmlByNodeName(XmlSchemaWrapper startingNode, string nodeName)

**XmlImportLogic.cs**

מחלקה סטאטית האחראית על ייבוא XML מוכן והפיכתו לאובייקט BL-י.

פונקציה לקריאת גרסת ה XML.

public static Version GetVersionOfXml(XmlDocument xmlDoc)

פונקציה לקריאת שם המשתמש שערך את ה XML.

public static string GetUserName(XmlDocument xmlDoc)

פונקציה לקריאת תאריך עריכת ה XML.

public static DateTime GetDateTime(XmlDocument xmlDoc)

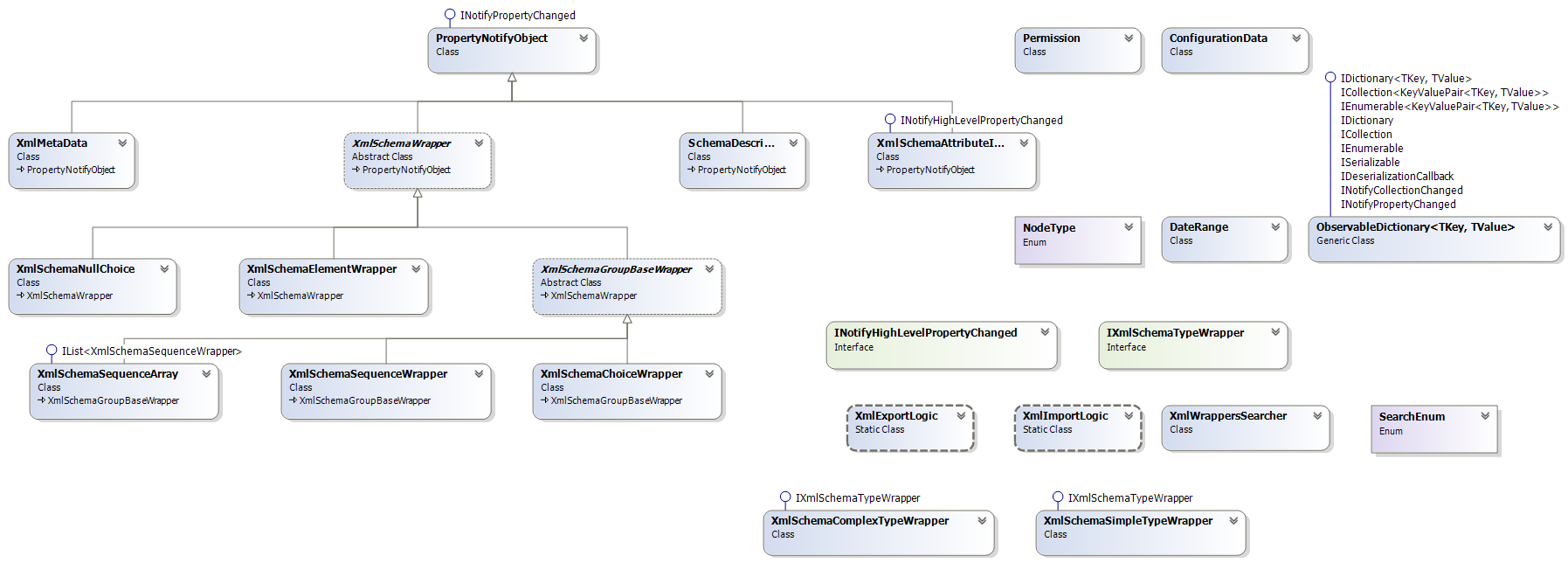
פונקציה לקריאת כלל המאפיינים של קובץ ה XML.

public static XmlMetaData GetAllProperties(XmlDocument xmlDoc, string path = "")

פונקציה להפיכת אובייקט ה XML לאובייקט BL-י, כלומר היררכיית אובייקטי ה Wrappers המקושרים לסכמה. מקבלת את האובייקט ה XML-י ואת השורש של הסכמה, וממלאת את השורש בתת עץ על פי ה XML הנקרא:

public static XmlSchemaElementWrapper XmlDocumentToSchemaWrapper(XmlDocument xmlDoc, XmlSchemaElementWrapper rootSchema)

דיאגרמת מחלקות עבור BL



# שכבת התצוגה - BobProject

בשכבה זו אנחנו משתמשים בממשק חלונאי כדי לייצור ממשק למשתמש עם המערכת.

שכבה זו משתמשת במידע המצוי בשכבת BL. הממשק למשתמש בנוי מחלון ראשי אחד שמכיל בתוכו תת חלוניות, כך שהמשתמש יכול לבחור באמצעות התפריט איזה חלון להציג.

***עקרונות מנחים***

* הארכיטקטורה המרכזית בשכבה זו היא מסוג MVVM (Model View ViewModel). האומר כי יש הפרדה מוחלטת בין View לבין הModel- כאשר התקשרות ביניהם היא באמצעות ViewModel הנעזרת במנגנוני Commands להעברת פקודות ומסרים ו-DataBinding להצגת הנתונים.
* עיקרון הכימוס (Encapsulation) בא לידי ביטוי בשכבה זו, בכך שהיא מסתיר את המידע הנשמר במחלקות וחושפת אותו כלפי חוץ באמצעות תכונות (Properties) בלבד.
* מחלקות מכילות מחלקות אחרות על מנת להימנע מירושה מסורבלת ולהרחיב יכולות, תוך שימוש ב Design Patterns שונים.

מחלקות חלונות (Views)

מחלקות אלו מייצגות חלונות שונים והן יורשות ממחלקה Window.

**Login.cs**

על מנת לנהל את פעולות המערכת, יש לדעת איזה סוג משתמש מחובר למערכת (מנהל מערכת / עורך / צופה). לשם כך, המחלקה הזו אחראית על כניסת משתמשים למערכת ומתן הרשאות.

*בנאי*

public Login()

מאתחל את החלון. קורא ממחלקת Permission את המשתמשים הפוטנציאליים ובודק האם חלה תקלה בעת הקריאה מהregistry. מציג הודעת שגיאה למשתמש במידת הצורך.

*פונקציות*

private void ShowSplashScreen()

private void Send\_Click()

private void Clear\_Click()

בחלון ישנן שני כפתורים עיקריים: שליחה ומחיקה. כאשר המשתמש ילחץ על כפתור שלח, המערכת תבדוק האם המשתמש נמצא במערכת ותבדוק את הרשאתו. כאשר המשתמש ילחץ על כפתור מחיקה אז המערכת תמחק את הנתונים שהוזנו בתיבות הטקסט.

אם המשתמש הזין שם משתמש וסיסמא נכונים, המערכת תפעיל את חלון SplashScreen.

**SplashScreen.cs**

כאשר המשתמש הזין שם משתמש וסיסמא נכונים, המערכת תציג את חלון ההמתנה אשר ברקע ייטענו המודלים של המערכת.

*בנאי*

public SplashScreen()

**MainWindow.cs**

זהו החלון הראשי של המערכת. חלון זה מכיל עץ של אובייקטים מסוג סכמה. כלומר, כל איבר בו הוא מסוג XmlSchemaWrapper. העץ מוצג בצורה היררכית. כאשר משתמש לוחץ על איבר כלשהו בעץ, המערכת מציגה את מאפייני האובייקט.

כאמור, מבנה העץ של סכמה יכול להיות מאוד ארוך ומסועף ולכן טעינת העץ היא לפי דרישה (Lazy load).

בנוסף, ניתן לראות בחלון מקרא צבעים עבור הטיפוסים השונים. התוכנה צובעת את העץ באמצעות המקרא הנ"ל. בתחתית החלון מוצג ההרשאה הנוכחית של המשתמש.

כאשר משתמש מזין ערכים במערכת, המערכת מציגה XML מעודכן.

בנוסף, ישנו תפריט שבאמצעותו ניתן לפתוח (בהתאם להרשאה): קובץ חדש, לייצא את המידע הקיים לקובץ XML , לצאת מהמערכת , לפתוח ולסגור את עץ האובייקטים, להראות את סרגל הכלים של החיפוש, להחליף משתמש, לפתוח את חלון הגדרות, לפתוח את חלון הדוחות ולפתוח את חלון אודות.

*בנאי*

private MainWindow()

המחלקה הינה singleton. מאתחלים בה את החלון וקוראים לפונקציה הפרטית Initialize.

*מאפיינים*

public MainWindowViewModel ViewModel { get; private set; }

public static MainWindow Instance

למחלקה יש מאפיין מסוג מופע המחלקה (המחלקה היא singleton) ומאפיין מסוג MainWindowViewModel המייצג את המקשר בין הview לmodel-.

*פונקציות*

פונקציה לאתחול החלון וביצוע binding בין פקדים למשתנים.

private void Initialize()

פונקציה הקוראת את המחרוזת משדה החיפוש ואת סוג החיפוש ומבצעת חיפוש במערכת. תוצאות החיפוש יוצגו על עץ האובייקטים.

private void OnSearch(object sender, RoutedEventArgs e)

פונקציה אשר מציגה את המאפיינים של תוצאת חיפוש הבאה.

private void ViewNextResult(object sender, RoutedEventArgs e)

פונקציה אשר מוחקת מהעץ את תוצאות החיפוש.

private void EraseAllLastSearchResult(object sender, RoutedEventArgs e)

פונקציה המציגה / מסתירה את סרגל הכלים של החיפוש.

private void ShowSearchBar(object sender, RoutedEventArgs e)

פונקציה אשר מרחיבה את כל העץ.

private void ExpandTree(object sender, RoutedEventArgs e)

פונקציה אשר מצמצמת את כל עץ.

private void CollapseTree(object sender, RoutedEventArgs e)

פונקציה אשר תופסת את האירוע שהעץ מתרחב. כאמור, מבנה העץ של סכמה יכול להיות מאוד ארוך ומסועף ולכן כאשר המשתמש מבקש לרדת ברמות העץ יש לטעון את השכבה הרלוונטית בעץ.

private void TreeViewItem\_Expanded(object sender, RoutedEventArgs e)

פונקציה אשר מייצרת את קובץ XML הזמני ומציגה אותו בפקד הרלוונטי*.*

public void UpdateXMLView()

**SaveXML.cs**

חלון שבאמצעותו מייצאים את המידע לקובץ XML. בחלון זה מזינים את שם העורך העוזר להפקת דוחות ונתיב הקובץ שאליו נייצא את הקובץ.

המערכת תבדוק כי קובץ XML שתפיק יהיה תקין, כלומר שכל הערכי חובה מולאו ע"י המשתמש. במידה וחלה שגיאה, תציג הודעה מתאימה.

*בנאי*

public SaveXML(SchemaDescriber schema)

מאתחלים בה את המסך.

*מאפיין*

public SaveXmlViewModel ViewModel { get; private set; }

מאפיין מסוג SaveXmlViewModel המייצג את המקשר בין הview לmodel-.

**Configuration.cs**

חלון המציג את הגדרות המערכת. ניתן לראות בו את מקרא צבעים עבור הטיפוסים השונים. המשתמש יכול לשנות עבור כל טיפוס צבע כרצונו. בנוסף, מוצג הנתיב של הסכמה שממנו המערכת קוראת נתונים.

רק בהרשאת מנהל יש אפשרות לשנות את נתיב הסכמה או/ו לשנות את הצבעים ולשמור בregistry לטעינה הבאה של התוכנית.

*בנאי*

public Configuration()

מאתחל את המסך.

*מאפיין*

public ConfigurationViewModel ViewModel { get; private set; }

מאפיין מסוג ConfigurationViewModel המייצג את המקשר בין הview לmodel-.

**Reports.cs**

חלון לניהול דוחות בהרשאת מנהל. בחלון זה ניתן לבחור את סוג החיפוש (לפי תאריך או/ו שם עורך) ואת התיקייה שממנה אנחנו רוצים לחפש והמערכת תציג את קבצי XML שנוצרו בהתאם לסוג החיפוש.

*בנאי*

public Reports(SchemaDescriber schema)

מאתחל את המסך.

*מאפיין*

public ReportsViewModel ViewModel { get; private set; }

מאפיין מסוג ReportsViewModel המייצג את המקשר בין הview לmodel-.

*פונקציה*

private void Browse\_Click(object sender, RoutedEventArgs e)

פונקציה המציגה חלון לבחירת תיקייה שתהווה תיקייה לחיפוש.

**About.cs**

חלון זה מציג את אודות המערכת הכולל: תיאור כללי של המערכת, כותבי התוכנה ומספר גרסה.

*בנאי*

public About()

מאתחל את המסך.

מחלקות כלליות

**ViewModelBase.cs**

מחלקה אבסטרקטית שכל המקשרים (ViewModel) יורשים ממנה. תפקידה לעלות אירוע כאשר המאפיין השתנה. ובכך הערך בGUI ישתנה באמצעות מנגנון ה-binding.

מחלקה היורשת ממשק INotifyPropertyChangedולכן ממששת את הפונקציות שלה.

*אירוע*

public event PropertyChangedEventHandler PropertyChanged;

*פונקציה*

protected void RaisePropertyChangedEvent(string propertyName)

מטודה המקבלת את שם המאפיין שהשתנה (propertyName) ומעלה את האירוע PropertyChanged על שם המאפיין שהשתנה.

**TreeViewHelper.cs**

מחלקה סטטית שתפקידה לטפל בעץ TreeView.

*פונקציות*

פונקציה סטטית המקבל עץ ומציגה את כל איבריו. כלומר, מרחיבה את העץ על העלים.

public static void ExpandAll(this TreeView treeView)

פונקציה סטטית המקבל עץ ומצמצמת את כל איבריו עד השורש.

public static void CollapseAll(this TreeView treeView)

פונקציה סטטית המקבל איבר בעץ ומרחיבה / מצמצמת אותו בהתאם לפרמטר isExpand.

public static void ExpandCollapseNode(this TreeViewItem item, bool isExpand)

פונקציה סטטית המקבל עץ, איבר לחיפוש (Item) וצבע וצובעת את האיבר הראשון בעץ השווה לitem בצבע המבוקש. מחזירה אמת אם נמצא האיבר. אחרת, שקר.

public static bool SelectItem(this TreeView treeView, object item, Color selectedColor)

פונקציה סטטית המקבל עץ, איבר לחיפוש (Item) וצבע וצובעת את כל האיברים בעץ השווים לitem בצבע המבוקש. ולבסוף מחזירה את כל האיברים שנצבעו כרשימה.

public static List<TreeViewItem> SelectItems(this TreeView treeView, object[] items, Color selectedColor)

**HierarchyDataTemplateSelector.cs**

כאמור, העץ מכילאובייקטים שונים מהסכמה (XmlSchemaWrapper). כל אובייקט בסכמה מוצג בעץ בצורה אחרת. לכן המחלקה יורשת מ- DataTemplateSelector ודורסת את הפונקציה SelectTemplate.

public override DataTemplate SelectTemplate(object item, DependencyObject container)

הפונקציה מקבלת איבר כלשהו בעץ. בודקת את סוג האיבר וממירה אותו לאובייקט כך שהGUI יידע את סוג האובייקט. כלומר הפונקציה יוצרת תבנית מותאמת לGUI.

**Splasher.cs**

מחלקת עזר סטטית לחלון SplashScreen. המחלקה מכילה מאפיין סטטי של החלון ופונקציות להצגת / הסתרת החלון.

public static Window Splash

public static void ShowSplash()

public static void CloseSplash()

**MessageListener.cs**

מחלקה סטטית סינגלטונית לשליחת הודעות. מטרתה לשלוח כי כל המודלים נטענו. המחלקה יורשת מ-DependencyObject.

*בנאי*

private MessageListener()

*מאפיינים*

public static MessageListener Instance

public string Message

*פונקציה*

public void ReceiveMessage(string message)

מקבלת הודעה ונעזרת ב- DispatcherHelper לשליחת אירוע מתאים.

**DispatcherHelper.cs**

מחלקה סטטית לשליחת אירועים.

*פונקציה שמייצרת אירוע לחלון* SplashScreen *שכל המודלים נטענו.*

public static void DoEvents()

מחלקות הקישור (ViewModel)

מחלקות אלו מייצגות את המקשרים השונים והן יורשות ממחלקה ViewModelBase.

**MainWindowViewModel.cs**

מחלקה אשר מהווה מקשר ביןModel לMainWindow. מכילה מאפיינים אשר יוצגו בGUI. על כל שינוי במאפיין תקרא לפונקציה RaisePropertyChangedEvent הנמצאת ב -.ViewModelBase

*בנאי*

public MainWindowViewModel()

אתחול המאפיינים והפקודות.

*מאפיינים*

מאפיין המייצג את הפקודה ShowPropertiesCommand אשר מציגה את הפרטים של אובייקט מהסכמה.

public ICommand ShowProperties { get; private set; }

מאפיין המייצג את הפקודה UpdateSeqNumItemsCommandשתפקידו להוסיף או להסיר איברים מ XmlSchemaSequenceArray.

public ICommand UpdateSeqNumItems { get; private set; }

מאפיין המייצג את הפקודה SelectedChoiceChangeCommand שתפקידו לטפל באירוע שהchoice השתנה באובייקט הסכמה.

public ICommand SelectedChoiceChange { get; private set; }

מאפיין המייצג את הפקודה DeleteSeqItemCommandשתפקידו לטפל במחיקת איבר ספציפי באובייקט XmlSchemaSequenceArray.

public ICommand DeleteSeqItem { get; private set; }

מאפיין המייצג את הפקודה LoadNewSchemaCommandשתפקידו לטפל בטעינת סכמה חדשה למערכת ופתיחת קובץ חדש (טעינה מחדש של הסכמה ומחיקת נתונים ישנים במערכת).

public ICommand LoadNewFile { get; private set; }

מאפיין המייצג את הפקודה ShowViewsCommandשתפקידו לטפל בהצגת מסכים לפי הארגומנט שנשלח.

public ICommand ShowViews { get; private set; }

מאפיין שתפקידו לייצג האם להציג את סרגל הכלים של החיפוש.

public bool IsShowSearchBar

מאפיין שתפקידו לייצג את שם ההרשאה הנוכחית במערכת.

public string Permit

מאפיין שתפקידו לייצג את אובייקט הסכמה שנבחר בעץ.

public XmlSchemaWrapper SelectedItem

מאפיין שתפקידו לייצג את מקרא הצבעים של המערכת.

public ObservableDictionary<string, Color> TypesColor

מאפיין שתפקידו לייצג בצורה היררכית את סכמת האובייקטים. העץ בGUI נעזר ברשימה זו.

public ObservableCollection<XmlSchemaWrapper> TypesList

*פונקציה*

פונקציה אשר טוענת את הסכמה במערכת.

public bool LoadSchema()

**SaveXmlViewModel.cs**

מחלקה אשר מהווה מקשר ביןModel ל SaveXML. מכילה מאפיינים אשר יוצגו בGUI. על כל שינוי במאפיין תקרא לפונקציה RaisePropertyChangedEvent הנמצאת ב -.ViewModelBase

*בנאי*

public SaveXmlViewModel(SchemaDescriber schema)

מקבל אובייקט מסוג SchemaDescriber ומאתחל את המאפיין SchemaDescriber. בנוסף, מאתחל את שאר המאפיינים והפקודות.

*מאפיינים*

מאפיין המייצג את SchemaDescriber הראשי של המערכת.

public SchemaDescriber SchemaDescriber { get; private set; }

מאפיין המייצג את נתיב הקובץ שאליו המערכת תייצא את קובץ XML.

public string PathFile

מאפיין המייצג את שם העורך שהמערכת תכתוב אותו בקובץ XML.

public string EditorName

מאפיין המייצג את הפקודה SelectXmlExportFileCommand שתפקידו להציג למשתמש חלון לבחירת נתיב ולעדכן את מאפיין PathFile.

public ICommand SelectXmlExportFile { get; private set; }

מאפיין המייצג את הפקודה ExportXmlCommand שתפקידו ייצא קובץ XML תקין. אחרת, להציג הודעת שגיאה מתאימה למשתמש.

public ICommand ExportXml { get; private set; }

**ConfigurationViewModel.cs**

מחלקה אשר מהווה מקשר ביןModel לConfiguration. מכילה מאפיינים אשר יוצגו בGUI. על כל שינוי במאפיין תקרא לפונקציה RaisePropertyChangedEvent הנמצאת ב -.ViewModelBase

*בנאי*

public ConfigurationViewModel()

מאתחל את המאפיינים והפקודות.

*מאפיינים*

מאפיין המייצג את הפקודה PickerColorCommand שתפקידו להציג למשתמש את חלון בחירת הצבעים ולעדכן את רשימת הצבעים בהתאם.

public ICommand PickerColor { get; set; }

מאפיין המייצג את הפקודה SaveConfigCommand שתפקידו לשמור את הקונפיגורציה החדשה (רשימת הצבעים והסכמה) – ניתן רק בהרשאת מנהל מערכת.

public ICommand SaveConfig { get; set; }

מאפיין המייצג את הפקודה LoadNewSchemaCommandשתפקידו לטפל בטעינת סכמה חדשה למערכת ופתיחת קובץ חדש (טעינה מחדש של הסכמה ומחיקת נתונים ישנים במערכת).

public ICommand LoadNewSchema { get; set; }

מאפיין המייצג את הצבע הנבחר ממקרא הצבעים.

public KeyValuePair<string, Color> SelectedItem { get; private set; }

מאפיין המייצג את אב של חלון Configuration.

public Window Parent { get; set; }

מאפיין המייצג האם המשתמש הנוכחי הוא בהרשאת "מנהל מערכת".

public bool IsManagerPerm { get; private set; }

מאפיין המייצג את מקרא הצבעים של המערכת.

public ObservableDictionary<string, Color> TypesColor

**ReportsViewModel.cs**

מחלקה אשר מהווה מקשר ביןModel לReports. מכילה מאפיינים אשר יוצגו בGUI. על כל שינוי במאפיין תקרא לפונקציה RaisePropertyChangedEvent הנמצאת ב -.ViewModelBase

*בנאי*

public ReportsViewModel(SchemaDescriber schema)

מקבל אובייקט מסוג SchemaDescriber ומאתחל את המאפיין SchemaDescriber. בנוסף, מאתחל את שאר המאפיינים והפקודות.

*מאפיינים*

מאפיין המייצג את SchemaDescriber הראשי של המערכת.

public SchemaDescriber SchemaDescriber { get; private set; }

מאפיין המייצג האם המשתמש בחר לחפש לפי שם עורך.

public bool IsEditorSearch

מאפיין המייצג האם המשתמש בחר לחפש לפי טווח תאריכים.

public bool IsDateSearch

מאפיין המייצג את התיקייה שממנה יחפשו את הקבצים.

public string PathFile

מאפיין המייצג את שם העורך שהמערכת תחפש את קבציו.

public string EditorName

מאפיינים המייצגים את טווח התאריכים שהמערכת תחפש ביניהם.

public DateTime StartDate

public DateTime EndDate

מאפיין המייצג את תוצאות חיפוש האחרונות אשר יוצגו ברשימה.

public ObservableCollection<XmlMetaData> LastSearchResults

מאפיין המייצג את הפקודה SearchReportsCommand שתפקידו לחפש את הקבצים לפי הפילוח שמשתמש בחר (לפי טווח תאריכים או/ו שם העורך).

public ICommand SearchFiles { get; private set; }

מחלקות הפקודות (Commands)

מחלקות אלו יורשות ממשק ,ICommand ולכן עליהן לממש את הפונקציות הבאות:

פונקציה המקבלת פרמטרים ובודקת האם הפקודה יכולה להתבצע.

public bool CanExecute(object parameter)

ופונקציה המבצעת את הפקודה.

public void Execute(object parameter)

**DeleteSeqItemCommand.cs**

תפקידו לטפל במחיקת איבר ספציפי באובייקט XmlSchemaSequenceArray.

*בנאי*

public DeleteSeqItemCommand(MainWindowViewModel \_viewModel)

מאתחל את הפקודה ושומר מאפיין של MainWindowViewModel על מנת לעדכן את פקד XML בהתאם.

*פונקציות*

ניתן לבצע את הפקודה במקרים הבאים:

* כאשר ההרשאה היא אינה "צופה".
* כאשר הפרמטר הוא מסוג XmlSchemaSequenceWrapper.
* כאשר האב האובייקט הוא XmlSchemaSequenceArray.

אם כל התנאים מתקיימים אז פונקציה CanExecute תחזיר אמת. אחרת, שקר.

public bool CanExecute(object parameter)

ביצוע הפקודה:

* המרת הפרמטר ל XmlSchemaSequenceWrapper.
* שליחת הודעה למשתמש לווידוא מחיקת האיבר.
* קריאה לפונקציה Remove של אב (Parent) XmlSchemaSequenceWrapper.
* עדכון פקד המציג את XML.

public void Execute(object parameter)

**ExportXmlCommand.cs**

תפקידו לייצא קובץ XML תקין. אחרת, להציג הודעת שגיאה מתאימה למשתמש.

*בנאי*

public ExportXmlCommand(SaveXmlViewModel \_viewModel)

מאתחל את הפקודה ושומר מאפיין של SaveXmlViewModel על מנת לבצע את הפקודה של ייצוא קובץ XML.

*פונקציות*

ניתן לבצע את הפקודה במקרים הבאים:

* כאשר כמות האלמנטים אינה ריקה.
* כאשר המאפיין SchemaDescriber.RootElement.AllChildAttributesFilled מחזיר אמת.

אם כל התנאים מתקיימים אז פונקציה CanExecute תחזיר אמת. אחרת, שקר.

public bool CanExecute(object parameter)

ביצוע הפקודה:

* בדיקה שמאפיין PathFile ב- SaveXmlViewModel אינו ריק ומכיל נתיב חוקי. אם לא, המערכת תציג הודעת שגיאה מתאימה.
* בדיקה שמאפיין EditorName ב- SaveXmlViewModel אינו ריק. אם לא, המערכת תציג הודעת שגיאה מתאימה.
* קריאה לפונקציה SchemaDescriber.ExportXmlNow ב SaveXmlViewModel עם נתיב הקובץ ושם העורך. במידה וחלה שגיאה, המערכת תציג הודעה בהתאם. אם הקובץ נוצר בהצלחה, המערכת תציג הודעה בהתאם.

public void Execute(object parameter)

**LoadNewSchemaCommand.cs**

תפקידו לטפל בטעינת סכמה חדשה למערכת ופתיחת קובץ חדש (טעינה מחדש של הסכמה ומחיקת נתונים ישנים במערכת).

*בנאי*

public LoadNewSchemaCommand(MainWindowViewModel \_viewModelMain)

מאתחל את הפקודה ושומר מאפיין של MainWindowViewModelעל מנת לבצע את הפקודה של טעינת סכמה / פתיחת קובץ חדש.

*פונקציות*

ניתן לבצע את הפקודה במקרים הבאים:

* כאשר הארגומנט הוא path"" וההרשאה הנוכחית במערכת היא מנהל מערכת.
* כאשר הארגומנט הוא ""local וההרשאה הנוכחית במערכת היא לא צופה.

אם כל התנאים מתקיימים אז פונקציה CanExecute תחזיר אמת. אחרת, שקר.

public bool CanExecute(object parameter)

ביצוע הפקודה:

* שליחת הודעה למשתמש שיידע כי פעולה זו מוחקת נתונים ישנים.
* בדיקה האם הארגומנט הוא "path".
  + אם כן, לפתוח חלון לבחירת סכמה חדשה.
  + טעינת סכמה באמצעות פונקציה LoadSchema שב-MainWindowViewModel.
  + עדכון התצוגה בחלונות מסך הגדרות ומסך הראשי.
* בדיקה האם הארגומנט הוא "local"
  + אם כן, מחיקת נתונים ישנים.
  + טעינת סכמה באמצעות פונקציה LoadSchema שב- MainWindowViewModel.
  + עדכון התצוגה בחלונות מסך הגדרות ומסך הראשי.

public void Execute(object parameter)

**PickerColorCommand.cs**

תפקידו להציג למשתמש את חלון בחירת הצבעים ולעדכן את רשימת הצבעים בהתאם.

*בנאי*

מאתחל את הפקודה ושומר מאפיינים של MainWindowViewModelו- ConfigurationViewModel על מנת לעדכן את מקרא הצבעים.

public PickerColorCommand(ConfigurationViewModel \_viewModelConf, MainWindowViewModel \_viewModelMain)

*פונקציות*

ניתן לבצע את הפקודה במקרים הבאים:

* כאשר המשתמש בחר צבע לשינוי ברשימת מקרא הצבעים.

אם כל התנאים מתקיימים אז פונקציה CanExecute תחזיר אמת. אחרת, שקר.

public bool CanExecute(object parameter)

ביצוע הפקודה:

* הצגת מסך לבחירת צבע.
* החלפת הצבע הנבחר בצבע הקודם ושמירתו ב MainWindowViewModelו- ConfigurationViewModel.
* עדכון תצוגה (עץ ומקרא).

public void Execute(object parameter)

**SaveConfigCommand.cs**

תפקידו לשמור את הקונפיגורציה החדשה (רשימת הצבעים והסכמה) – ניתן רק בהרשאת מנהל מערכת.

*בנאי*

מאתחל את הפקודה.

public SaveConfigCommand()

*פונקציות*

ניתן לבצע את הפקודה במקרים הבאים:

* כאשר ההרשאה של המשתמש היא מנהל מערכת.

אם כל התנאים מתקיימים אז פונקציה CanExecute תחזיר אמת. אחרת, שקר.

public bool CanExecute(object parameter)

ביצוע הפקודה:

* קריאה לפונקציה SaveConfig של ConfigurationData.
* הצגת הודעה למשתמש במצב של הצלחה / כישלון.

public void Execute(object parameter)

**SearchReportsCommand.cs**

תפקידו לחפש את הקבצים לפי הפילוח שמשתמש בחר (לפי טווח תאריכים או/ו שם העורך).

*בנאי*

public SearchReportsCommand(ReportsViewModel \_viewModel)

מאתחל את הפקודה ושומר מאפיין של ReportsViewModel על מנת לבצע חיפוש קבצים לפי הפילוח.

*פונקציות*

ניתן לבצע את הפקודה במקרים הבאים:

* כאשר ההרשאה של המשתמש היא מנהל מערכת.
* כאשר נתיב החיפוש אינו ריק.

אם כל התנאים מתקיימים אז פונקציה CanExecute תחזיר אמת. אחרת, שקר.

public bool CanExecute(object parameter)

ביצוע הפקודה:

* בדיקה איזה סוג חיפוש יש לבצע (לפי טווח תאריכים או/ו שם העורך).
* קריאה לפונקציה SchemaDescriber.ProduceReport שב ReportsViewModel-.
* עדכון תצוגה – רשימת התוצאות.

public void Execute(object parameter)

**SelectedChoiceChangeCommand.cs**

תפקידו לטפל באירוע שהchoice השתנה באובייקט הסכמה.

*בנאי*

public SelectedChoiceChangeCommand(MainWindowViewModel \_viewModel)

מאתחל את הפקודה ושומר מאפיין של MainWindowViewModelעל מנת לעדכן את התצוגה (עץ האובייקטים).

*פונקציות*

ניתן לבצע את הפקודה במקרים הבאים:

* כאשר SelectedItem שב MainWindowViewModelאינו ריק.

אם כל התנאים מתקיימים אז פונקציה CanExecute תחזיר אמת. אחרת, שקר.

public bool CanExecute(object parameter)

ביצוע הפקודה:

* קריאה ל DrillOnce ל-SelectedItem.
* עדכון התצוגה (עץ האובייקטים).

public void Execute(object parameter)

**SelectXmlExportFileCommand.cs**

תפקידו להציג למשתמש חלון לבחירת נתיב ולעדכן את מאפיין PathFile שב- SaveXmlViewModel.

*בנאי*

public SelectXmlExportFileCommand(SaveXmlViewModel \_viewModel)

מאתחל את הפקודה ושומר מאפיין של SaveXmlViewModel על מנת לבחור נתיב חוקי לשמירת קובץ XML.

*פונקציות*

ניתן לבצע את הפקודה במקרים הבאים:

* כאשר ההרשאה של המשתמש אינה צופה.

אם כל התנאים מתקיימים אז פונקציה CanExecute תחזיר אמת. אחרת, שקר.

public bool CanExecute(object parameter)

ביצוע הפקודה:

* פתיחת חלון לשמירת קובץ.
* בחירת נתיב ע"י המשתמש.
* עדכון מאפיין PathFile ב- SaveXmlViewModel.

public void Execute(object parameter)

**ShowPropertiesCommand.cs**

מציגה את הפרטים של אובייקט מהסכמה.

*בנאי*

public ShowPropertiesCommand(MainWindowViewModel \_viewModel)

מאתחל את הפקודה ושומר מאפיין של MainWindowViewModel על מנת להציג את הפרטים של אובייקט הסכמה.

*פונקציות*

ניתן לבצע את הפקודה במקרים הבאים:

* כאשר האובייקט XmlSchemaWrapper אינו ריק.

אם כל התנאים מתקיימים אז פונקציה CanExecute תחזיר אמת. אחרת, שקר.

public bool CanExecute(object parameter)

ביצוע הפקודה:

* המרת הפרמטר לאובייקט מסוג XmlSchemaWrapper.
* בדיקת סוג האובייקט (מאפיין NodeType).
* אם האובייקט הוא Element אז האובייקט הוא מסוג XmlSchemaElementWrapper.
  + אםIsSimple אמת אז יש להציג את InnerText.
  + אם Attributes.Count > 0 אז יש להציג את המאפיינים שלו.
* אחרת, אם האובייקט הוא Choice אז האובייקט הוא מסוג .XmlSchemaChoiceWrapper ולכן יש להציג combobox עם כל האפשרויות.
* אחרת, אם האובייקט הוא Sequenceאז האובייקט הוא מסוג XmlSchemaSequenceArray. ולכן יש להציג numericUpDown עם מספר איברי המערך.
* אחרת, אין מאפיינים.

public void Execute(object parameter)

**ShowViewsCommand.cs**

תפקידו לטפל בהצגת מסכים לפי הארגומנט שנשלח.

*בנאי*

public ShowViewsCommand(MainWindowViewModel \_viewModel)

מאתחל את הפקודה ושומר מאפיין של MainWindowViewModel על מנת להציג את חלונות השונים תחת חלון הראשי.

*פונקציות*

ניתן לבצע את הפקודה במקרים הבאים:

* כאשר הארגומנט הוא אחד ממחרוזות הבאים: "Login"/ / "Configuration" "Reports"/ "SaveXML"/ . "About"

אם כל התנאים מתקיימים אז פונקציה CanExecute תחזיר אמת. אחרת, שקר.

public bool CanExecute(object parameter)

ביצוע הפקודה:

* המרת הארגומנט למחרוזת ובדיקה איזה מסך רוצים להציג.
* Owner של החלון החדש הוא החלון הראשי (MainWindow).
* אם רוצים לפתוח את מסך SaveXML אז יש לבדוק כי כל השדות החובה של עץ האובייקטים מולאו. אם לא, יוצג הודעת שגיאה למשתמש.

public void Execute(object parameter)

**UpdateSeqNumItemsCommand.cs**

תפקידו להוסיף או להסיר איברים מ XmlSchemaSequenceArray.

*בנאי*

מאתחל את הפקודה.

public UpdateSeqNumItemsCommand()

*פונקציות*

ניתן לבצע את הפקודה במקרים הבאים:

* כאשר הארגומנט הוא מסוג XmlSchemaSequenceArray.

אם כל התנאים מתקיימים אז פונקציה CanExecute תחזיר אמת. אחרת, שקר.

public bool CanExecute(object parameter)

ביצוע הפקודה:

* המרת הארגומנט לאובייקט מסוג XmlSchemaSequenceArray.
* בדיקת כמות האיברים החדשה שהוזנה ע"י המשתמש.
* בדיקה האם הכמות החדשה גדולה / קטנה / שווה לכמות האיברים המצויה בזיכרון.
* אם הכמות גדלה אז להוסיף את האיברים ע"י קריאה איטרטיבית לפונקציה AddNewWrapper.
* אם הכמות קטנה אז להודיע למשתמש כי האיברים האחרונים הם יימחקו ולמחוק את האיברים מהסוף ע"י קריאה לפונקציה RemoveAt.
* עדכון תצוגה בהתאם.

public void Execute(object parameter)

מחלקות המרת נתונים (ValueConverters)

מחלקות אלו יורשות ממשק , IValueConverter ולכן עליהן לממש את הפונקציות הבאות:

פונקציה המקבלת ערך X ומחזירה לערך חדש Y.

public object Convert(object value, Type targetType, object parameter, CultureInfo language)

ופונקציה הממירה להפך, הכוונה שהפונקציה מקבלת ערך Y וממירה בחזרה לערך X.

public object ConvertBack(object value, Type targetType, object parameter, CultureInfo language)

בפרויקט זה לא נממש את פונקציה ConvertBack.

**BooleanToVisibilityConverter.cs**

פונקציה המקבלת ערך (value) בוליאני וממירה לראות. אם הערך הוא אמת אז הפונקציה תחזיר "הצג". אחרת תחזר "הסתר".

public object Convert(object value, Type targetType, object parameter, CultureInfo language)

**ColorToBrushConverter.cs**

פונקציה המקבלת ערך צבע (אובייקט Color) וממירה אותו ל-SolidColorBrush.

public object Convert(object value, Type targetType, object parameter, CultureInfo language)

**ElementsToVisibilityConverter.cs**

פונקציה המקבלת ערך אובייקט מסוג XmlSchemaElementWrapper ופרמטר מסוג מחרוזת. המחרוזת יכולה להיות "Attributes" / " ."Inner

אם הארגומנט הוא "Attributes" ו- Attributes.Count > 0 אז החזר "הצג".

אם הארגומנט הוא "Inner" ו-מאפיין IsSimple הוא אמת אז החזר "הצג".

אחרת, החזר "הסתר".

public object Convert(object value, Type targetType, object parameter, CultureInfo language)

**IndexToTextConverter.cs**

פונקציה המקבלת ערך מספר, מגדילה אותו ב-1. ומחזירה את התוצאה כמחרוזת.

public object Convert(object value, Type targetType, object parameter, CultureInfo language)

**PermissionToEnableConverter.cs**

פונקציה המקבלת ערך מחרוזת המייצג את ההרשאה הנוכחית ופרמטר מסוג מחרוזת. המחרוזת יכולה להיות " Editor" / " ." Manager

אם ההרשאה הנוכחית היא מנהל מערכת והפרמטר הוא Manager אז החזר אמת.

אחרת, אם ההרשאה הנוכחית היא עורך או מנהל מערכת והפרמטר הוא Editor אז החזר אמת.

אחרת, החזר שקר.

public object Convert(object value, Type targetType, object parameter, CultureInfo language)

**PermissionToVisibilityConverter.cs**

פונקציה המקבלת ערך מחרוזת המייצג את ההרשאה הנוכחית ופרמטר מסוג מחרוזת. המחרוזת יכולה להיות " Editor" / " ." Manager

אם ההרשאה הנוכחית היא מנהל מערכת והפרמטר הוא Manager אז החזר "הצג".

אחרת, אם ההרשאה הנוכחית היא עורך או מנהל מערכת והפרמטר הוא Editor אז החזר "הצג".

אחרת, החזר "הסתר".

public object Convert(object value, Type targetType, object parameter, CultureInfo language)

**SelectedItemToArrayConverter.cs**

פונקציה המקבלת ערך מסוג אובייקט XmlSchemaWrapper. וממירה את האובייקט ל- ObservableCollection<XmlSchemaWrapper>. ומחזירה את התוצאה.

public object Convert(object value, Type targetType, object parameter, CultureInfo language)

**SelectionPropertyToVisibilityConverter.cs**

פונקציה המקבלת ערך מסוג NodeType המייצג את סוג אובייקט הסכמה ופרמטר מסוג מחרוזת. המחרוזת יכולה להיות " Element" / " " Sequence" / " Choice/ " Null".

אם NodeType הוא Element והפרמטר הוא Element אז החזר "הצג".

אחרת, אם NodeType הוא Choice והפרמטר הוא Choice אז החזר "הצג".

אחרת, אם NodeType הוא Sequenceוהפרמטר הוא Sequence אז החזר "הצג".

אחרת, אם NodeType הוא NULLאו SequenceItem והפרמטר הוא NULL אז החזר "הצג".

אחרת, "הסתר".

public object Convert(object value, Type targetType, object parameter, CultureInfo language)

**ColorLegendConverter.cs**

מחלקה זו שונה מקודמותיה. היא יורשת מ IMultiValueConverter. ועלייה לממש את הפונקציות הבאות:

פונקציה מקבלת 2 ערכים: אובייקט מסוג XmlSchemaWrapper ואת מקרא הצבעים.

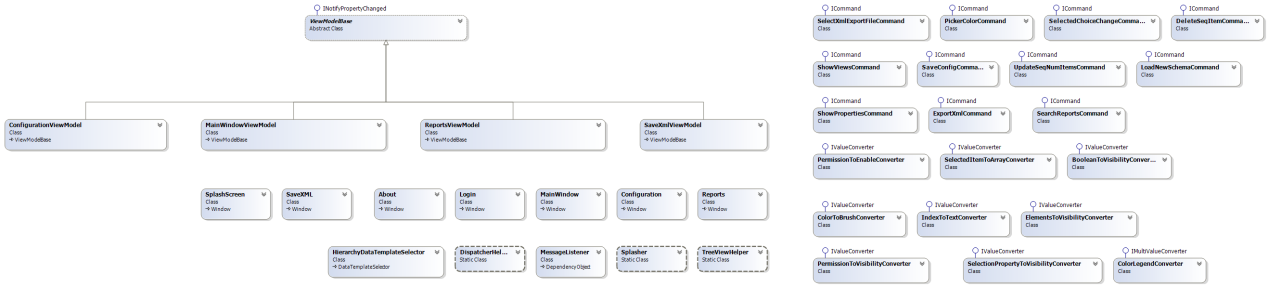
הפונקציה בודקת את סוג האובייקט לפי NodeType. ובהתאם לכך היא מחזירה את הצבע המתאים לפי המקרא. הפונקציה מחזירה אובייקט מסוג SolidColorBrush.

public object Convert(object[] value, Type targetType, object parameter, System.Globalization.CultureInfo culture)

הפונקציה הבאה לא נממש.

object[] IMultiValueConverter.ConvertBack(object value, Type[] targetTypes, object parameter, System.Globalization.CultureInfo culture)

דיאגרמת מחלקות עבור GUI



**דיאגרמות רצף:**

ייצוא קובץ XML

****חיפוש דוחות

****חיפוש טיפוס או ערך

טעינת סכמה

**בדיקת האם הסכמה תקינה**

****

**שימוש ברכיבי שפה ודפוסי עיצוב**

* שימוש במנגנון ה-events של C#. לדוגמא במחלקה ViewModelBase ובמחלקה PropertyNotifyObject אשר מפיצות events למחלקות האחרות.
* שימוש בפולימורפיזם. לדוגמא: במחלקת XmlSchemaWrapper יש רשימה של אובייקטים בשם Children מסוג XmlSchemaWrapper. אשר יכולים להיות כל אובייקט היורש מ XmlSchemaWrapper.
* מחלקות אבסטרקטיות. לדוגמא: מחלקה XmlSchemaWrapper ומחלקה XmlSchemaGroupBaseWrapper.
* העמסת מתודות. לדוגמא: במחלקת XmlSchemaWrapper ישנו מאפיין בשם AllChildAttributesFilled וכל מחלקה אשר יורשת ממחלקה XmlSchemaWrapper. דורסת את המאפיין וממששת מחדש. (כמו מחלקת XmlSchemaChoiceWrapper).
* שימוש בThreading. לדוגמא: במסך SplashScreen ליצירת "מד התקדמות" בטעינת המודלים.
* שימוש בתבנית ארכיטקטורה  MVVM (Model View ViewModel) בשכבת התצוגה (GUI). ארכיטקטורה  זו אומרת כי יש הפרדה מוחלטת בין View לבין הModel- כאשר התקשרות ביניהם היא באמצעות ViewModel הנעזרת במנגנוני Commands להעברת פקודות ומסרים ו-DataBinding להצגת הנתונים.
* שימוש בדפוס עיצוב Singleton. לדוגמא: מחלקת ConfigurationData, מחלקה Permission ומחלקת MainWindow.
* שימוש בדפוס עיצוב Composite. לדוגמא: במחלקת XmlSchemaComplexTypeWrapper.
* שימוש בדפוס עיצוב Factory Method. לדוגמא: במחלקת XmlSchemaGroupBaseWrapper.
* שימוש בדפוס עיצוב Decorator . לדוגמא: במחלקת XmlSchemaWrapper.

**שיפורים ושינויים עתידים**

המערכת תוכננה כך ששינויים עתידיים יהיו קלים ליישום:

1. שמירת הגדרות: הגדרות המערכת נשמרות בRegistry. ניתן לשנות את זה ולשמור את ההגדרות בקובץ חיצוני (לדוגמא XML).
2. יכולות החיפוש מתקדמות: להוסיף אפשרות לחפש בתכונה ספציפית עם ערך כלשהו, שאילתות בסגנון SQL.
3. עריכת XML ידנית והשפעה על תצוגת העץ: כרגע ה XML הוא לתצוגה בלבד. ניתן לאפשר למשתמש לערוך אותו ישירות ולראות איך הדבר משפיע על העץ ההיררכי.
4. טעינת מספר סכמות במקביל: כרגע ניתן לטעון סכמה אחת בלבד בזמן נתון. ניתן לאפשר לטעון מספר סכמות, לשמור את נתיביהן ואז להחליף ביניהן ואף לאפשר ולידציה ע"י מספר סכמות ביחד.
5. יצירת XML לדוגמה ע"י סמכה: אפשרות לייצר קובץ מלא לדוגמה עם טעינת הסכמה, כך שהמשתמש יראה XML מלא שלפיו יוכל לערוך את ה XML שלו.
6. ניהול XML-ים: המערכת תאפשר מסך עם רשימה של כל ה XML-ים, כולל מאיזו סכמה נוצרו ופתיחתם דרכו (ולא רק דו"ח). בנוסף, ניתן להוסיף קישור בין XML-ים, כאלו שמשתפים טיפוסים ביניהם.
7. ייצוא ממשק: מפתחים עתידיים יוכלו להתחבר למערכת BOB ע"י ממשק (DLL או Web Service) ובכך לדלות מידע אודות סכמות ו XML-ים שהמערכת ייצרה.