**Remote Client Management**



**תיק פרוייקט**

מיכאל קשקט (321479081)

אלכס ליפוב ()

**מבוא**

תוכנת Remote Client Management היא תוכנה לניהול תחנות קצה המחוברות דרך LAN (באותה רשת), WAN (סניפים מרוחקים) או באצמעות אינטנרט (לפטופים ניידיים המחוברים לקוו אינטרנט לו דווקא מהיר). התוכנה מחזיקה מאגר תחנות שפעם אחד התחברו אליה, ומציגה סטטוס של כל תחנה (מחוברת\מנותקת) וכן את המידע הבסיסי שלהן: מס' סידורי של מחשב, נתונים על מע' הפעלה, רשת, כמות זיכרון פיזי, שטח דיסק ועוד..

כמו כן לתחבנות המחוברות התוכנה מאפשרת:

* לקבל רשימת תוכנות המותקנות במחשב
* לנתר (להציג real-time גרפים): תפוסת מעבד, שטח פנוי בדיסק סיסטם (שבו נמצאת מע' הפעלה), עומס על הדיסק (ע"י הצגת תור ממוצע של כתיבות\קריאות), זיכרון פיזי פנוי.
* הרצת תהליכים בתחנה המרוחקת, כולל הרצת תוכנות \ התקנות מרוחקותעם delay, שמאפשר לתכנן הרצות שלא בשעות עבודה.
* נעילה, אתחול וכיבוי מחשב מרוחק.
* להדליק מחשב כבוי מרוחק (רק בתוך LAN\WAN, בתנאי שבאותו subnet יש כבר מחשבים מנוהלים דלוקים)
* הרצת ping ו- traceroute ממחשב מרוחק לכל יעד שהוא.
* אפשרות להשתלט על מחשב המרוחק (שימוש בתוחנה זעירה צד שלישי - VNC – שמגיעה עם התוכנה ועובדת באופן שקוף למשתמש).

התוכנה בעלת 3 רכיבים:

* שרת מרכזי (מתוכנן שירוץ בתור service, אך בגלל הרשאות מוגבלות בכיתת מחשב – רץ כקונסולה)
* תוכנת לקוח (רצה ב-System Tray)
* ממשק ניהול מרכזי (בעלת GUI פשוט ונח, רצה מכל מחשב מרוחק (בדרך כלל של אדמיניסטרטור))

**דרישות מערכת:**

* Windows XP/2003/Vista/7/2008
* .Net Framework 4 ומעלה
* 10Mb שטח דיסק ועד 50Mb RAM

**הוראות התקנה:**

התוכנה מגיעה כתיקיה אחד שבתוכה יש 4 תת-תיקיות:

1. "Server" - תוכנת שרת ותוכנת ניהול
2. "Agent" - תוכנת לקוח קצה
3. "Console" – ממשק ניהול
4. "Common" – תיקיה משותפת לכל הרכיבים הנ"ל.

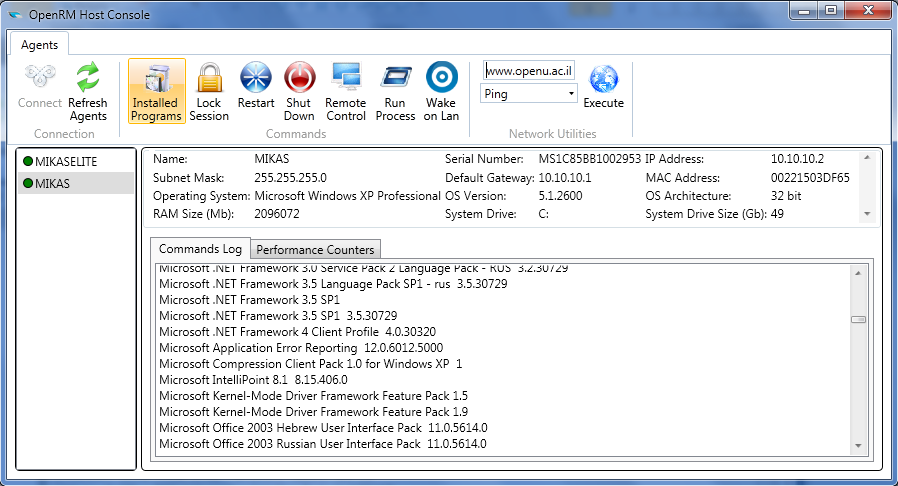
התקנה של רכיב תוכנה כלשהו מורכב משתי שלבים:

1. העתקה תיקית הרכיב ואת התיקיה "Common" לתיקיה כלשהו במחשב (למשל, בתחנות קצה מנוהלות יהיו שתי תיקיות בלבד: "Agent" ו- "Common", אחת ליד שנייה).
2. הגדרת IP של שרת בקובץ קונפיגורציה של כל רכיב (לא רלוונטי לשרת עצמו).

קבצי קונפיגורציה הינם קצבים ששמם מסתיים ב-"config". האופציות שניתנות לקינפוג:

* שרת:
  + "ListenOnPort" – פורט שנאזין עליו
  + "MaxConnections" – כמות מקסימלית של חיבורים (במקביל). ברירת מחדל – 2000. מעל כמות כזאת מומלץ לעשות בדיקות נוספות.
  + "LogFilePattern" – שם של קובץ לוג. "[date]" בתוך השם הזה אוטומטית יוחלף לתאריך הרצה.
  + "Secret" – סיסמא בעלת 2^x (חזקה של 2) תווים שחייבת להיות זהה בכל רכיבי התוכנה (אחרת – רכיב לא יצליח לעבוד עם שרת)
* רכיב לקוח וממשק ניהול:
  + "ServerIP" – IP של שרת
  + "ServerPort" – פורט של שרת
  + "LogFilePattern" – שם של קובץ לוג. "[date]" בתוך השם הזה אוטומטית יוחלף לתאריך הרצה.
  + "Secret" – סיסמא שחייבת להיות זהה בכל רכיבי התוכנה (אורך הסיסמא – חזקה של 2).

**הוראות שימוש:**

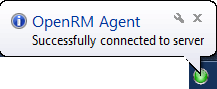
****

הפעלת תוכנה:

1. יש לוודא ש-IP של שרת בקבצי קונפיגורציה מעודכן וזהה בכל המחשבים.
2. יש להפעיל שרת ע"י כניסה לתיקיה "Server" והרצת קובץ EXE.

הערה: בגלל שבתכנון תהליך של שרת אמור לרוץ כ-service, אין לו ממשק גרפי. כל הדברים נכתבים ל-log, וזה בהחלט מספק. לכן הוא נפתח בחלון "dos". יש פשוט להקטין את החלון ולא להתייחס אליו.

1. הפעלת client: יש להיכנס לתיקית "Agent" ולהפעיל קובץ EXE. ב-System Tray של מחשב יופיע עיגול אדום. יש ללחוץ עליו עם כפתור ימני של עכבר, ובתפריט שנפתח לבחור "Start OpenRM Agent". עיגול יחליף צבע לירוק, ומיד יופיע notification שבו יהיה כתוב אם הצליח להתחבר לשרת:



הערות: במידה ובזמן הזה השרת לא היה פעיל, Agent יתריע על כך (תופיע notification) וינסה להתחבר לבד אחרי כמה שניות. הוא ינסה להתחבר שוב ושוב, עד שלא יצליח או עד שיכבו אותו (בתפריט יבחרו "Stop…" ו-icon יחליף צבע לאדום). האינטרוול בין ניסיונות יילך ויגדל ב-5 שניות, ויגיע ל-60 שניות לכל היותר.

במידה ורוצים להתחבר לשרת אחר, ללא שינוי בקובץ קונפיגורציה, נכנסים ל-"Settings" בתפריט של Agent ומחליפים IP או פורט, ולוחצים "Apply". חשוב לזכור שהשינוי הזה לא יישמר בקובץ קונפיגורציה! כלומר, הוא תקף עד לכיבוי של Agent.

1. הפעלת ממשק ניהול: יש להיכנס לתיקית "Console" ולהפעיל קובץ EXE.

ייפתח חלון GUI. כדי להתחבר לשרת יש ללחוץ על כפתור "Connect". אם מצליח – יהפוך כפתור "Connect " ללא פעיל, וכפתור "Refresh Agents" יהפוך לפעיל. אם לא מצליח להתחבר – תינתן הודעה למסך.

יש ללחוץ על כפתור "Refresh Agents": ברגע זה נקבל מהשרת רשימה של כל ה-Agents שהשרת מכיר.



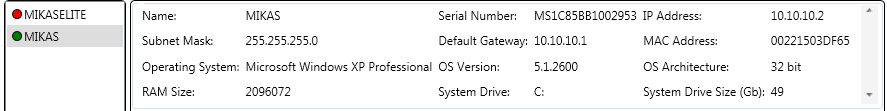
הערה: ניתן להשתמש בממשק ניהול אחד בלבד עבור שרת.

**תפעול:**

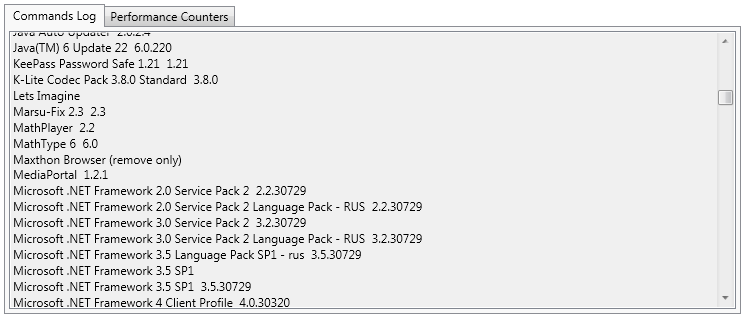
כל פעולות מתבצעות ממשק הניהול המותקן בתחנה של אדמיניסטרטור. אדמיניסטרטור שולח בקשות לשרת, עם ציון תחנת קצה שעליו יש לבצע עבודה, ורכיב הלקוח רק עונה לבקשות השרת.

פעולות שאדמינסטרטור יכול לבצע על מחשב מרוחק:

1. לראות נתונים סטטיים על המחשב בלחיצה על השם שלו ברשימה בצד שמאל:



1. להביא רשימת תוכנות המותקנות במחשב המרוחק בלחיצת כפתור . הרשימה מתקבלת בטאב "Commands Log" של החלון הראשי:



1. לנעול, לאתחל או לכבות מחשב בעזרת כפתורים:

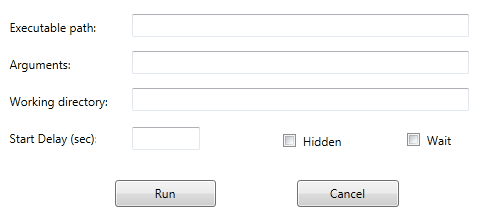


1. להשתלט על המחשב בלחיצת כפתור:



בזמן שבוחרים בהשתלטות, בתחנת ניהול עולה VNC client, ובתחנה המנוהלת עולה VNC server, והוא פותח connection ל-VNC client (ולא הפוך, כי אם בתחנה המרוחקת יש windows firewall – ההשתלטות תיכשל). חשוב לזכור: התוכנה עולה באופן שקוף (אמנם היא מופיעה ב-SystemTray), אבל בגלל שזאת תוכנה "צד שלישי" שעובדת על פורט אחר, firewall יכול "לצעוק" עליה.

1. להריץ פקודה\תוכנה כלשהי הנמצאת על המחשב המרוחק: בלחיצה על הכפתור  נפתח חלון שבתוכו מזינים פרטי התוכנה, פרמטרים והשהיה לפני ההרצה (אם ההרצה היא מיידית – שמים 0):

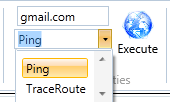


הסבר: "Executable Path" – נתיב אבסולוטי לקובץ הרצה, "Hidden" מריץ תהליך מוסתר ממשתמש, "Wait" מחכה לסיום התהליך ושולח החזרה את קוד היציאה (exit code, כדי לדעת האם הוא יצא בשגיאה, ומה היא השגיאה).

1. ניתן להעיר תחנה כבוייה לחיצה על כפתור  ("WOL"). לכל מחשב שנוצר בשנים אחרונות יש כבר תמיכה במנגנון הזה, אבל הפעולה עדיין יכולה להיכשל בגלל המגבלה: המנגנון מתבסס על שליחה Broadcast, לכן כדי שהפעולה תצליח אנחנו חייבים שבותו subnet יהיה לפחות agent אחד פעיל (אז הוא ישלח את ה-broadcast הזה). השרת בוחר כזה agent באופן שרירותי.

הערה: הכפתור של "WOL" פעיל רק ל-agent-ים לא מחוברים.

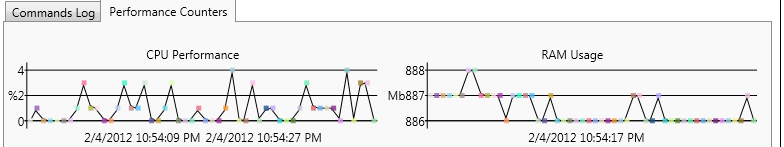
1. ניתן להריץ ping או traceroute מתחנה המרוחקת בחלק המסך הבא:



יש להזין את הכתובת הרצוייה, וללחוץ על "Execute". את תוצאת הפעולה ניתן לראות בשדה "Commands Log".

חשוב לזכור: לפעולה TraceRoute בדרך כלל לוקח כמה שניות לסיים, לכן צריך להיות טיפה סבלני: נראה את התוצאה רק בסיום הפעולה (לאומת tracert של windows שמציג פלט על כל תחנה שבדרך).

1. ניתן לקבל גרפים של ביצועים (real-time) ע"י מעבר לטאב "Performance Counters":



נקודות חשובות לתפעול המערכת:

זיהוי של תחנה, בזמן ההתחברות שלה לשרת, מתבצע ע"י שם מחשב ומס' סידורי.

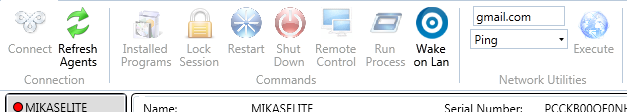
שרת שומר נתונים (סטטיים) על כל תחנה שאי פעם התחברה אליו. זאת על מנת להחזיק מאגר תחנות ברשת, ולדעת לשלוח Wake-On-Lan לתחנה כבוייה.

השרת יכול לבקש גם לשלוח מחדש עדכון של נתונים סטטיים שיש לו כבר, כיוון שהם סטטיים עד לכיבוי המחשב (למשל, כמות הזיכרון יכולה להשתנות כיוון שלקחו מחשב למעבדה והוסיפו לו זיכרון).

כל כמה זמן השרת שמור את הנתונים הסטטיים האלה לדיסק, לכן להוריד אותו בלי לדאוג לנתונים.

לאחר השבתת השרת ועלייתו בחזרה, כל ה-agents יתחברו אליו שוב, תוך דקה לכל היותר.

אם תחנה (client) מתנתק משרת, נשלח עדכון לממשק הניהול, הוא נצבע באדום ואי אפשר לשלוח אליו יותר בקשות, כי כל הכפתורים הופכים להיות דהויים (כולם חוץ מ-WOL):



הערה: אם תחנה התנתקה מהשרת אבל בממשק הניהול היא עדיין מופיעה כ-online, וננסה להפעיל פעולה כלשהי עליה - השרת לא יטפל בבקשה וישלח עדכון סטטוס התחנה לממשק הניהול.

אם יש טעות בסיסמא ("secret" בקובץ הגדרות), השרת יסגור connection כבר בניסיון ראשון לשליחת מסר כלשהו, בלי "התראה מוקדמת מראש".

**ארכיטקטורת תוכנה:**

המערכת בנוייה ארבעה רכיבים (המהווים Solution אחד בתוך VS):

1. שרת ("Host"):

ללא ממשק.

לב המערכת. מחזיק נתונים על כל ה-Agent-ים. שולח אליהם פקודות שרובם מקבל ממשק הניהול, מקבל מהם תשובה שאותה מעבריר לממשק הניהול.

פותח בפרוייקט בשם "OpenRm.Server.Host".

1. ממשק ניהול ("Gui"):

GUI מבוסס WPF.

מדבר רק עם שרת, לא ישירות עם Agent-ים.

מחולק לשלושה פרוייקטים: " OpenRm.Server.Gui.App" (ראשי), "OpenRm.Server.Gui.Modules.Monitor" (מכיל את כל התצוגות) ו- "OpenRm.Server.Gui.Inf" (עזרים).

1. תוכנת לקוח ("Agent"):

בעל ממשק "קל": Icon ב-System Tray, עם כמה אופציות בלבד.

מקבל פקודות מהשרת, מבצע אותם, ושולח תשובות בחזרה (אם יש צורך).

מחולק לשני פרוייקטים: " OpenRm.Agent" (המנוע עצמו), ו-"OpenRm.Agent.CustomControls" (ממשק).

1. ספריות משותפות:

מכיל את האובייקטים המשותפים לכל רכיבי המערכת: רכיבי תקשורת עיקריים, הודעות, רכיבי המרה, הצפנה, עזרים שונים...

נמצאים בפרוייקט בשם "OpenRm.Common.Entities".

כלומר, יש סה"כ 3 מודולים, ו-Library משותף.

**שיקולים שהיו בתכנון, צורת העבודה של המערכת:**

משתמשים ב-TCP פרוטוקול, בעיקר בגלל אמינותו: כל המודולים מדברים בינים בעזרת הודעות – אובייקטים שעברו סריאליזציה למחרוזת (xml). לכן חשבו גם סדר הגעת הפקטות, וגם תוכנן שלהם. עוד סיבה לשימוש ב-TCP: יכולת לזהות תחנות לא פעילות (כבויות \ מתנתקות וכדומה).כדי למקסם את היכולת הזאת, שרת שומר connection פתוח מול כל הלקוחות שהתחברו אליו, וכל פרק זמן נשלח keep-alive (בשני הכיוונים, כיוון שגם client צריך לדעת אם צד שני יתנתק באופן לא יזום).

בחרנו ללמוד ולהשתמש ב- "High Performance TCP Sockets" החדשים (יחסית): SocketAsyncEventArgs (שמנו הסבר על זה במצגת). זה נותן לשרת יכולת לעבד ולהחזיק אלפי חיבורים (בתאוריה), לעבד הרבה בקשות של client-ים לא יציבים (שמתנתקים כל הזמן). וכמובן שזה נותן ביצוע פעולות שליחה\קבלה בצורה א-סנכרונית.

היה חשוב ליישם מנגנון שבו שליחת הודעות לא תהיה ביחס 1-1 לקבלה. כמלומר, שלא יווצר מצב שבו אחרי שליחת הודעה נצטרך לחכות לקבלת תשובה, כדי שנוכל לשלוח הודעה הבאה. פתרנו את הבעיה ע"י הקצאת 2 אובייקטים SocketAsyncEventArgs עבור connection: אחד תמיד מוכן לקבל מידע, ובשני משתמשים בזמן השליחה.

SocketAsyncEventArgs מייצרים thread-ים (באופן שקוף), ובמידת הצורך אנחנו יוצרים thread-ים נוספים, לכן היה חשוב לזכור לדאוג שקוד יהיה "thread-safe" עבור אובייקטים משותפים.

טיפול במגבלות של קבלת הודעות של TCP מתבסס על הוספת Prefix בתחילת כל הודעה.

כל המודולים של המערכת מדברים בעזרת העברת אובייקטים יעודיים (מסוג "Message") שמייצגים פקודות ותשובות (סוגים: "RequestBase" ו-"ResponseBase", הנמצאים תחת OpenRm.Common.Entities -> Network -> Messages).

האובייקטים האלה עוברים סריאליזציה לפורמט XML, בעזרת Woxalizer (במקום מובנה של .NET) – dll צד שלישי שפותח כ-open-source עבור שפות שונות. זה נותן לנו בעתיד לפתח agent-ים ב-JAVA עבור מע' הפעלה Linux, Android ואחרים.

בגלל ש-xml – טקסט פשוט, וההודעות שלנו יכולות להכיל מידה חשוב \ סודי, אבל עוברות ברשת אינטרנט ציבורית - לכן כל הודעה מוצפנת בעזרת אלגוריתם סינכרוני חזק ידוע - AES. בגלל שמפתחות ההצפנה לא מועברים בערוץ לא מאובטח (אלא נמצאים בקבצי קונפיגורציה ובתוך קוד עצמו) – למי שיאזין לתעבורה ברשת אין סיכוי לפנח אותה. כמובן קיימת אופציה לבטל את ההצפנה (לצורך debug, לדוגמה) ע"י הזנת סיסמא ריקה ("") בקובץ קונפיגורציה.

אם מנסים להתחבר לשרת בלי לדעת את המפתחות - השרת ינתק קשר. כל פעם ששולחים אליו מידע לא תקין - הוא ינתק קשר. זה נותן איזה שהוא מנגנון להתגונן מפריצות.

השרת מאזין לפורט TCP יחיד (גם עבור כל ה-client-ים, גם עבור ממשק ניהול), זה משיקולים אסטטיים ואבטחת מידע.

ברגע ש-Agent מתחבר, שרת שלוח לו בקשה להזדרות (לא בלבל עם "אימות"): Agent שולח לו את השם ומס' הסידורי שלו. השרת עושה השוואה מול הנתונים שיש לו כבר: אם הוא רואה אותו פעם ראשונה, הוא מקצה לו ID ושומר אצלו ב-"database" (במבנה מסוג Dictionary). לאחר מכן השרת מבקש לשלוח אליו נתונים סטטים בסיסיים על מע' הפעלה וחומרה, וברגע שמקבל אותם – שמור גם כן ב-"database". כך הוא שומר inventory של כל התחנות, וגם מציג מידע סטטי לממשק ניהול בצורה יותר מהירה.

...

**אובייקטים ופונציות עיקריות:**

TcpServerListenerAdv.cs ומחלקת אב שלה TcpBase.cs:

כאן נמצאים כל המטודות שדרושות להקמת שרת TCP:

Init: בניית pool של SocketAsyncEventArgs-ים, הקצאת מקום בזיכרון עבורם (buffers)

Start: הקמת TCP socket שמאזין לפורט ספציפי

StartAccept: הקצאת אובייקט SocketAsyncEventArgs לקבלת מתחברים חדשים, וקריאה אסנכרונית של AcceptAsync.

ProcessAccept: הקצאת שני אובייקטים מסוג SocketAsyncEventArgs מתוך pool שבנינו: אחד - לשליחה, שני – קבלה.

StartReceive: קריאה לקבלה אסנכרונית ReceiveAsync (ואם הקריאה האסנכרונית נכשלת – קוראים ל-ProcessReceive בצורה סנכרונית).

ProcessReceive: טיפול בקבלת הודעות: קוראים מתוך buffer של אובייקט מסוג SocketAsyncEventArgs שאחראי על הקבלה. קודם כל בודקים את גודל ההודעה בעזרת prefix . ברגע שיודעים את גודל ההודעה, מנסים לקרוא את הכמות הזאת של ביתים: קוראים כמה שניתן מ-buffer . אם לא מספיק – שוב קוראים לפונקציה StartReceive, בצורה רקורסיבית. אחרי קבלת הודעה שלמה – מנקים משתנים זמניים. אם נשאר משהו ב-buffer שלא "השתמשנו" בו – זאת הודעה באה, לכן ממשיכים בתהליך כמו קודם. אם אם יותר נתונים - מתחילים להאזין שוב לנתונים חדשים.

MessageSend: הכנת הודעה: הוספת prefix, שמירת הודעה בתוך UserToken של אובייקט המוגדר כשולח. כמו כן אנו מונעים משליחה כפולה\מקבילית באותו ערוץ (לאותו client ואותו אובייקט מסוג SocketAsyncEventArgs) ע"י semaphore. זאת כדי לשמור על שלמות ההודעות (וחוץ מזה, SendAsync לא מאפשר לשלוח שתי הודעות במקביל).

StartSend: התחלת שליחת הודעה: הכנת buffer (העתקה כמה שניתן תווים לתוך buffer), וקריאה ל-SendAsync.

ProcessSend: טיפול בשליחת הודעה: אם ההודעה ארוכה מידי (גדולה מ-buffer) – קוראים ל-StartSend בצורה רקורסיבית: כל פעם שולחים חלק בגודל שקבענו מראש. בסיום השליחה של הודעה שלמה משחררים semaphore, ואז אם יש פעולת Send שממתינה – היא מתחילה לשלוח.

CloseConnection: סגירת socket, ניקוי אובייקטים אסנכרוניים של שליחה וקבלה, חזרתם ל-pool.

ProcessReceivedMessage: עיבוד הודעה מלאה שהתקבלה (פענוח, דה-סריאליזציה, חזרה ל-callback).

ProcessFailure: טיפול בכישלון בשליחה\קבלה (סגירת connection, דיווח ל-callback).

GeneralSocketClient.cs:

המחלקה ביחד עם TcpBase.cs מהווים TCP מודול של Agent ושל ממשק ניהול.

אותו עיקרון כמו של שרת, רק לא מחזיקים Pool של אובייקטים אסנכרוניים, ולא Buffer גדול.

WoxalizerAdapter.cs: המרת אובייקט ל-xml (לזיכרון או לדיסק), ובחזרה.

EncryptionAdapter.cs: רכיב הצפנה ופענוח.