Digital Whisper
Technologic Papers

# איסוף זבל ב-.NET

נכתב ע"י סשה גולדשטיין

## הקדמה

בהמשך למאמר בגיליון חודש יוני, אנו מתקדמים בסקירתנו את המימוש הפנימי של .NET. ובמאמר זה נבחן את פעולתו של אוסף הזבל (garbage collector) ומבנה הערימה המנוהלת (GC heap) שעליה הוא פועל. כידוע, איסוף הזבל הוא המאפיין המרכזי של סביבות מנוהלות כגון Java ו-.NET. ומקל משמעותית על המתכנת כיוון שכעת ביכולתו להתעלם מהצורך בשחרור מפורש של אובייקטים מהזיכרון.

לאוסף הזבל חשיבות מכרעת לביצועים גבוהים ביישומי צד שרת, משחקים רגישים לעיכוב, מערכות זמן אמת ועוד רבות אחרות. לכן אופן פעולתו מורכב יחסית, כדי לאפשר ביצועים גבוהים לסוגים שונים של תוכניות. אם התקורה של איסוף הזבל עולה על התקורה של ניהול ידני של זיכרון, זה עשוי להיות לא משתלם חרף הרווח הגדול ביעילות הפיתוח.

נתחיל בסקירה כללית של פעולת אוסף הזבל בשיטת סימון ומחיקה (mark and sweep) ולאחר מכן נכנס לפרטים. עוד לפני שנתחיל יש צורך לציין שאוסף הזבל של .NET. אינו פועל בשיטת מניית התייחסויות (reference counting), שאינה מקובלת באופן כללי באוספי זבל של שפות עילית מודרניות. תוכלו לקרוא עוד על מניית התייחסויות בוויקיפדיה וכן לעיין בפרטים אודות אוסף הזבל של Python, המשתמש בין היתר במניית התייחסויות.

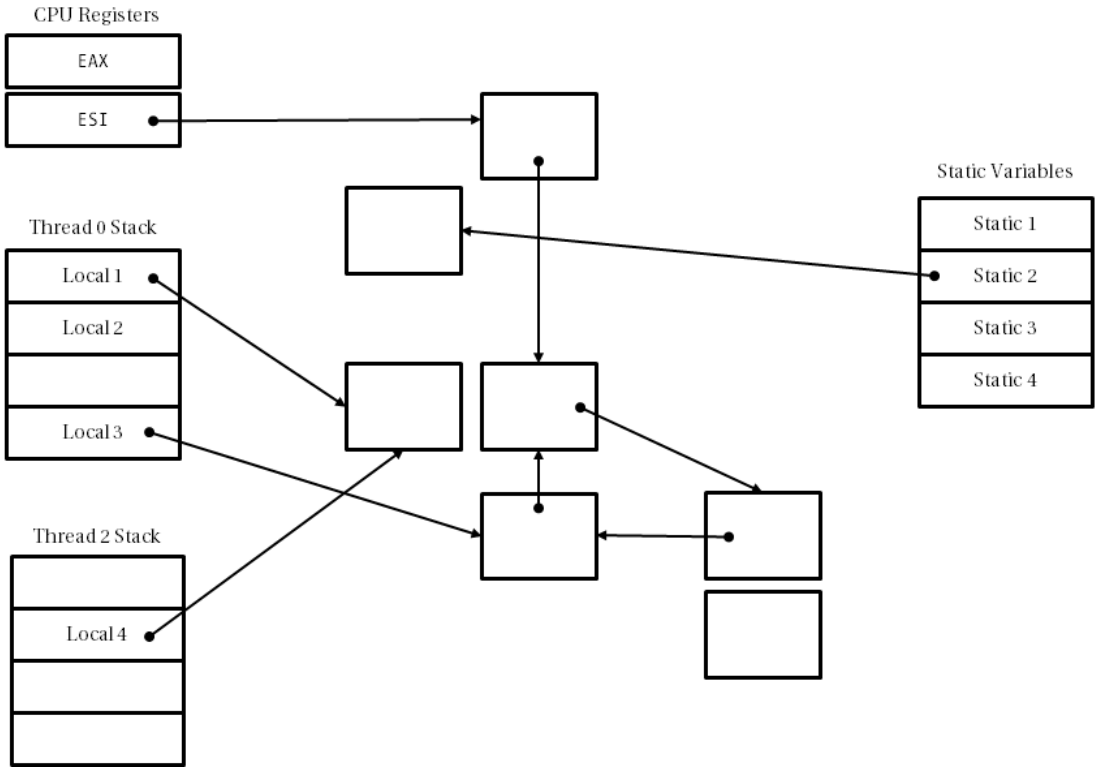
## סימון ומחיקה

העיקרון הבסיסי העומד ביסוד פעולתו של אוסף הזבל הוא מושג הישיגות (reachability). אוסף הזבל נדרש להשאיר בערימה את האובייקטים הישיגים, אלה שהתוכנית עשויה להשתמש בהם בהמשך, ולמחוק מן הערימה את שאר האובייקטים (הזבל). ההגדרה של ישיגות היא רקורסיבית, ונובעת מהאופן שבו תכניות יכולות לגשת לאובייקטים באופן כללי:

1. אובייקט הוא ישיג אם יש מצביע אליו ממשתנה סטטי של מחלקה טעונה כלשהי.
2. אובייקט הוא ישיג אם יש מצביע אליו ממשתנה מקומי של שיטה המתבצעת כרגע.
3. אובייקט הוא ישיג אם יש מצביע אליו מאוגר (register) של מעבד כלשהו המבצע את התוכנית.
4. אובייקט הוא ישיג אם יש מצביע אליו מאובייקט ישיג.

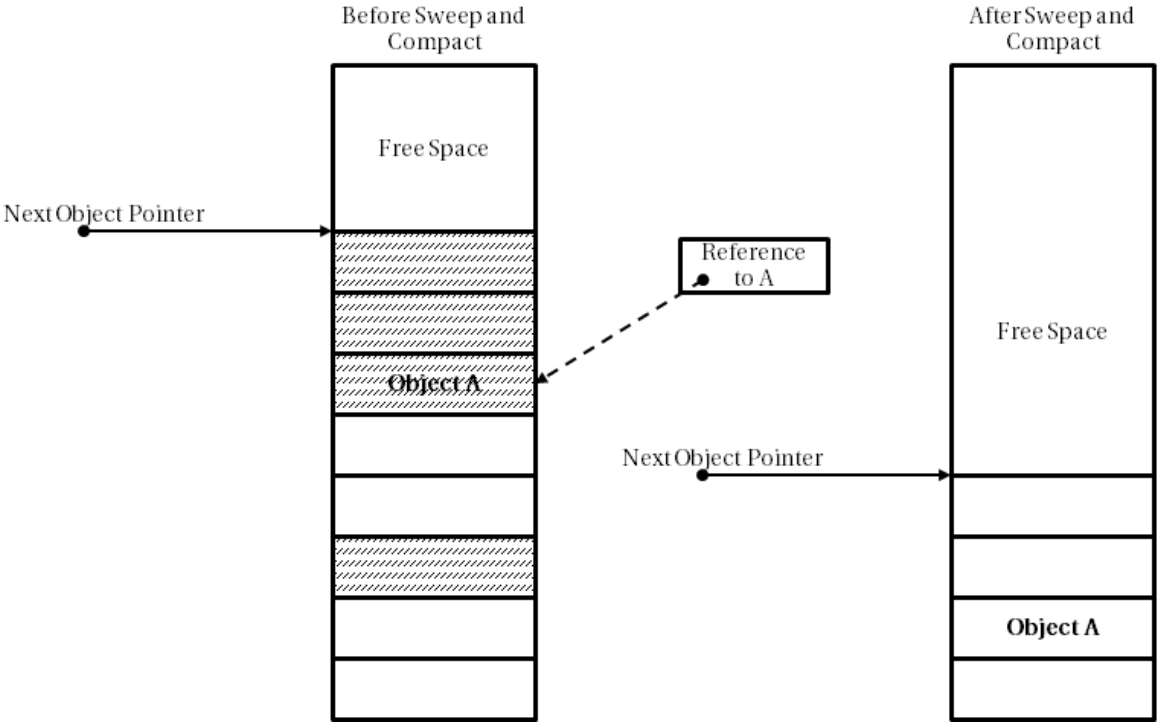
יש לשים לב שהגדרת ישיגות זו רחבה יותר מאשר "האובייקטים שהתוכנית משתמשת בהם". יתכן שמשתנה סטטי מסוים מכיל מצביע לאובייקט שלא ייעשה בו שימוש עד סוף ריצת התוכנית. עם זאת, אוסף הזבל לא יכול לקבוע האם התוכנית תעשה שימוש במצביע מסוים (הדבר שקול לבעיית העצירה, שאינה כריעה על ידי מחשב), ולכן הגדרת ישיגות זו היא האופן שבו פועלים כל אוספי הזבל בשיטת סימון ומחיקה.

אם כן, כדי למצוא את האובייקטים המיועדים למחיקה (הזבל), על אוסף הזבל למצוא תחילה את האובייקטים הישיגים. הדבר מתבצע על ידי חיפוש רגיל בגרף, תוך סימון האובייקטים הישיגים ב- object header word שהוזכר במאמר הקודם. נקודות ההתחלה של החיפוש הם המשתנים הסטטיים של כל המחלקות הטעונות, והמחסניות של החוטים הפעילים שבהן נמצאים המשתנים המקומיים של שיטות המתבצעות כרגע. בניגוד לאוספי זבל מסוימים אחרים, אוסף הזבל של NET. הוא מדויק ביחס למשתנים מקומיים: יש ברשותו תמיד מידע עדכני על המשתנים המקומיים של כל שיטה, ולא ניתן להכשילו על ידי משתנים שאינם מכילים מצביעים אך שנראים כמו כאלה (למשל, float שערך הביטים שלו נראה כמו מצביע לתוך הערימה).




בסוף שלב הסימון אוסף הזבל פונה למחיקת האובייקטים הלא-ישיגים. על מנת לשמור על רציפות הזיכרון ולקבל ביצועים טובים יותר בזמן הקצאת אובייקטים, אוסף הזבל של NET. מבצע מעת לעת דפרגמנטציה של הערימה, על ידי דחיסת האובייקטים הישיגים למקטע זיכרון רצוף (מה שכרוך בהזזה של אובייקטים ישיגים ועדכון המצביעים אליהם כדי שימצאו אותם במקומם החדש). לאחר פעולה זו, ניתן לחלק את

הערימה למקטע שמכיל אך ורק אובייקטים ישיגים ומקטע שהוא פנוי לחלוטין, וכך עלות ההקצאה מהערימה היא אפסית כמעט וכרוכה רק בקידום מצביע המכונה next object pointer, בדיוק כמו הקצאת זיכרון מהמחסנית באמצעות ה-stack pointer.




## עצירת חוטים

כאשר אוסף הזבל מתחיל במעבר על הערימה המנוהלת, נשאלת השאלה: האם שינויים המתבצעים ע"י חוטים אחרים בתוכנית ועלולים להביא לתוצאות איסוף לא מדויקות? נראה שכך המצב, ונדמה שאין ברירה אלא לעצור את כל החוטים של התוכנית בכל מהלך פעולתו של אוסף הזבל. לגישה זו, המכונה stop-the-world, חיסרון עצום: בפועלו על ערימה גדולה (של מספר ג"ב), אוסף הזבל עשוי לעצור את כל התקדמות התוכנית למספר שניות ואף דקות! יתר על כן: עצירה של חוטים אחרים בצורה אכזרית (SuspendThread ודומיו) מסוכנת מאוד, ובפועל אוספי זבל נוקטים בשיטה המבוססת על שיתוף פעולה בין חוטי התוכנית לבין החוט של אוסף הזבל. שיתוף פעולה זה יקר מאוד ומכניס עיכובים נוספים (עד מאות מ"ש) הדרושים רק לשם עצירת החוטים בהכנה לתהליך האיסוף.

אלא שכבר מגרסתו הראשונה, אוסף הזבל של NET. מאפשר עצירה חלקית של החוטים האחרים בתוכנית. למעשה, אוסף הזבל מאפשר בחירה באיסוף בו-זמני (concurrent), הדורש עצירה של החוטים האחרים

בעיקר במהלך שלב המחיקה אך לא בשלב הסימון. אובייקטים חדשים המתווספים לערימה במהלך הסימון ושינויי מצביעים לאובייקטים במהלך שלב הסימון דורשים התייחסות מיוחדת של אוסף הזבל, כדי שלא לפנות אובייקטים הנמצאים עדיין בשימוש התוכנית.

בגרסאות האחרונות של .NET, אוסף הזבל השתכלל בהרבה בכל הקשור לביצוע איסוף במקביל לפעולת התוכנית. ראשית, כברירת מחדל אוסף הזבל פועל היום בתצורה בו-זמנית הן בצד השרת והן בתחנות עבודה. אוסף הזבל מפעיל מספר חוטים הרצים ברקע ומבצעים איסוף זבל כמעט ללא עצירה של חוטי התוכנית, והאוספים את הזבל במקביל תוך ניצול ריבוי הליבות (או המעבדים) הקיים היום כמעט בכל מחשב. על אף שאוסף הזבל של .NET עדיין נמצא מאחור ביחס למוצרים כגון Azul Pauseless GC עבור JVM, כבר היום משך עצירת החוטים והתקורה הנגרמת מאיסוף זבל המתרחש לעתים קרובות הצטמצמו משמעותית ומספקים את הדרישות אפילו של מערכות זמן אמת מסוימות (למערכות רגישות באמת .NET מאפשרת גם לבקש לצמצם את פעולת אוסף הזבל באמצעות API מפורש המכונה GCLatencyMode.)

## דורות

אחת הסיבות לעצירות ממושכות של המערכת בזמן איסוף הזבל היא הצורך לעבור על כמות גדולה של אובייקטים בזיכרון. בתוכנית המשתמשת באופן קבוע ב-1 ג"ב של זיכרון ומקצה לעתים רחוקות אובייקטים חדשים, כל סיבוב איסוף זבל מחייב סימון של כל האובייקטים הישיגים בערימה. אפילו אם כל האובייקטים אכן נמצאים בזיכרון הפיזי זוהי פעולה ממושכת יחסית (מאות מ"ש), ואם מקצת מהאובייקטים דופדפו לקובץ ההחלפה (pagefile), זמן זה יכול להאמיר לשניות ארוכות. כדי לצמצם עלויות אלה, אוסף הזבל מבחין בין אובייקטים חדשים לאובייקטים ותיקים, ומנצל שתי תכונות ניסיוניות שבדרך כלל מתבררות כנכונות לגבי רוב התוכניות:

1. אובייקטים חדשים מתים מהר.
2. אובייקטים ותיקים נוטים להישאר בחיים זמן ממושך.

תכונות דומות ידועות בתחום מערכות ההפעלה לגבי זמני ריצה של תהליכים - תהליכים שרצו כבר זמן ממושך צפויים להמשיך לרוץ עוד זמן רב, בעוד שתהליכים שנוצרו זה עתה צפויים להסתיים בקרוב. בתחום מערכות ההפעלה נעשה שימוש בתכונות אלה על מנת לבצע תזמון חכם של תהליכים; בתחום איסוף הזבל לתכונות אלה חשיבות מכרעת בחלוקת הערימה לאזורי איסוף נפרדים (דורות).

אוסף הזבל של .NET מחלק את הערימה לשלושה דורות (generations) - דור 0 לאובייקטים חדשים לגמרי, דור 1 לאובייקטים פחות חדשים, ודור 2 לאובייקטים ותיקים. המעבר בין הדורות מתרחש כאשר מתבצע איסוף זבל - אובייקט בדור 0 ששרד סיבוב איסוף אחד יעבור לדור 1, אובייקט בדור 1 ששרד איסוף נוסף יעבור לדור 2. אלא שבבואו לבצע איסוף זבל, אוסף הזבל לא מסמן את כל האובייקטים הנמצאים בערימה, אלא רק את האובייקטים בדור המיועד לאיסוף (condemned generation) והדורות שמתחתיו.

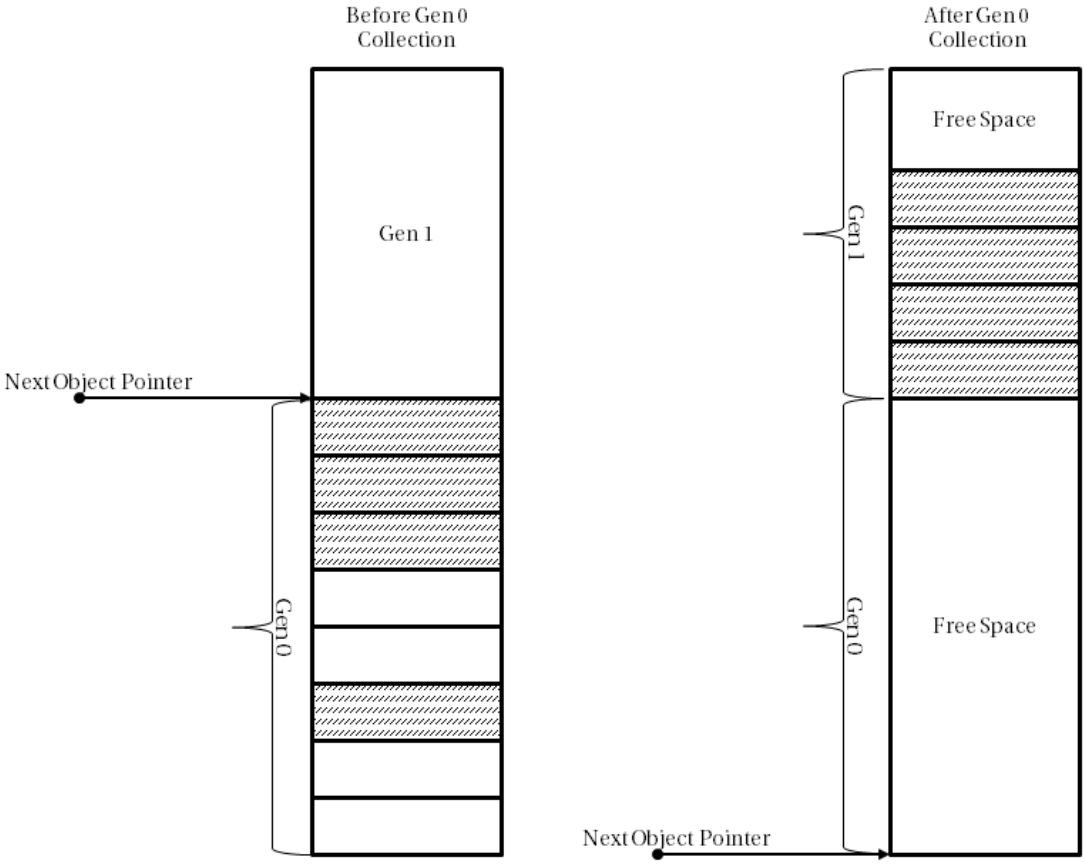
כך למשל, כאשר נגמר המקום בדור 0 (שגודלו מצומצם מאוד ויכול להיות מספר מ"ב בודדים) אוסף הזבל יבצע סבב סימון ומחיקה בדור 0 בלבד. זוהי פעולה זולה מאוד מאחר שדור 0 בדרך כלל נכנס בזיכרון המטמון של המעבד, וממילא מדובר על כמות קטנה יחסית של אובייקטים שיש לסמן. האובייקטים השורדים יועברו לדור 1 ופעולת האיסוף תסתיים במהרה. באופן דומה, כאשר נגמר המקום בדור 1, מתבצע איסוף זבל בדור 1 - וכדומה.

**על שמירת סיסמאות בזיכרון**

בדומה לדפים המוקצים ישירות בעזרת VirtualAlloc שמגיעים לתוכנית מלאי אפסים (zero pages), אובייקטים או חוצצים המוקצים ב-.NET. נדרסים באפסים לפני שהתוכנית יכולה לעשות בהם שימוש. אלא שלאיסוף הזבל חיסרון מסוים, כיוון שאין לדעת מתי הזיכרון שאינו נמצא בשימוש יוחזר למערכת ההפעלה או ינוקה מתוכן. בהחלט ייתכן שמידע רגיש שנשמר בחוצץ או מחרוזת בזיכרון לא יימחק ולא יידרס באפסים במשך שעות ארוכות.

על מנת להתמודד חלקית עם אתגר זה, .NET מספקת מחלקה הנקראת SecureString המבטיחה שתווי המחרוזת נשמרים בזיכרון בצורה מוצפנת בלבד. הדבר אינו מונע חשיפה של המידע כיוון שגם מפתח ההצפנה נמצא בזיכרון, וישנם גם API-ים מפורשים להמרה של SecureString למחרוזת רגילה (כגון Marshal.SecureStringToBSTR), אך מדובר בהגנה ראשונית בפני דליפת מידע לא מכוונת במסגרת התוכנית. דוגמא לשימוש:

```
ConsoleKeyInfo key;
var pwd = new SecureString();
while(true) {
  key = Console.ReadKey(true);
  if (key.Key ==
      ConsoleKey.Enter) break;
  pwd.AppendChar(key.KeyChar);
}
Process.Start("calc.exe",
  "Administrator", pwd, null);
```




התועלת הגדולה משיטת האיסוף הדורות עולה מהתבוננות בהנחות (1) ו-(2) שצוינו מעלה. כאשר אוסף הזבל מסמן ומנקה את דור 0, רבים הסיכויים שרוב האובייקטים בדור 0 אינם ישיגים ולכן ניתנים לפינוי. כלומר, למרות שדור 0 קטן ויתרחשו בו סבבים תכופים של איסוף זבל, אלה יהיו סבבים יעילים ומהירים כיוון שרוב האובייקטים ניתנים למחיקה ולא דורשים עבודה נוספת מצד אוסף הזבל.

המשמעות העיקרית עבור תכניות היא החשיבות בשמירה על הנחות (1) ו-(2) לעיל. כלומר, על תכניות מנוהלות לבצע הקצאות ארוכות טווח בתחילת הריצה ולא לשחרר לעתים קרובות אובייקטים ותיקים, ומאידך לדאוג לשחרור מהיר ככל האפשר של אובייקטים חדשים (זמניים) כדי שיוכלו להיאסף עוד בדור 0. מניסיון רב שנים בתחום עולה שתוכניות הפועלות לפי הנחיות אלה משיגות שיפור של מספר סדרי גודל בזמני איסוף הזבל, ומגיעות למצב בו איסוף הזבל כמעט ואינו מורגש ברקע ריצת התוכנית.

## הקצאת זיכרון ממערכת ההפעלה

מהדיון קודם בניהול הערימה ע"י אוסף הזבל מתבהר שלעתים קרובות מקטעים רחבים של הערימה פנויים מאובייקטים לחלוטין. עם זאת, אוסף הזבל לא מחזיר אותם מידית למערכת ההפעלה, ולמעשה אוסף הזבל גם לא פונה למערכת ההפעלה עבור כל הקצאה של אובייקט חדש - משיקולי ביצועים. אוסף הזבל אף אינו משתמש במנגנוני ניהול הערימה של חלונות, כגון HeapAlloc/HeapFree, אלא פונה ישירות למנהל הזיכרון הווירטואלי באמצעות VirtualAlloc.

אוסף הזבל מבקש ממערכת ההפעלה מקטעי זיכרון גדולים המכונים סגמנטים, שגודלם נע בין 64-16 מ"ב רצופים במערכות 32 ביט ועד מספר ג"ב רצופים במערכות 64 ביט. מדובר בשיריון כתובות בלבד, שמתהווה להקצאה של זיכרון ממש רק כאשר נעשה בדפי הזיכרון שימוש.


על ידי בקשה של מקטעי זיכרון גדולים, אוסף הזבל מצמצם את התקורה הכרוכה בפנייה לשירותים של מערכת ההפעלה (system calls) וכן בניהול טבלאות תרגום של כתובות וירטואליות לכתובות פיזיות ( page tables). עם זאת, לשיטת הסגמנטים גם חיסרון משמעותי, בעיקר במערכות 32 ביט שבהן מרחב הכתובות מוגבל מאוד.

במערכת הפועלת שעות וימים ארוכים ומבצעת כמות משמעותית של הקצאות הן מהערימה המנוהלת והן ממקורות אחרים במערכת ההפעלה (כגון טעינה של DLL-ים או הקצאה של זיכרון מהערימה הלא-מנוהלת), עשויה להיווצר פרגמנטציה חמורה של מרחב הכתובות הוירטואליות, עד כדי כך שהקצאות לא תצלחנה למרות שקיימים עדיין דפים פנויים רבים. תופעה זו היא אחת הסיבות למנגנוני מחזור (recycling) אוטומטיים המופעלים היום כמעט בכל יישום צד-שרת, והדואגים להפעיל את התהליך מחדש לאחר מספר

שעות/ימים כדי לקבל מרחב כתובות "טרי" ללא פרגמנטציה. לאבחון של תופעות פרגמנטציה מסוג זה, היישום VMMap מבית Sysinternals מועיל מאוד (ובאמצעותו נוצר האיור הקודם).

## סיכום

במאמר זה סקרנו את אופן פעולתו של אוסף הזבל ב-.NET, ונחשפנו מעט לשיקולים העומדים בבסיס מימושו הפנימי. למרות תחרות קשה מצד אוספי זבל מסחריים אחרים, והתקדמות מהירה בתחום גם מן המישור האקדמי, אוסף הזבל של .NET מצליח לענות על דרישותיהן של רוב היישומים המנוהלים ומצדיק את היתרון העצום ביעילות הפיתוח שמספקות השפות המנוהלות.

קצרה היריעה מכדי להיכנס לעומק בכל הקשור לאיסוף זבל ב-.NET, ולמעשה ראינו רק את קצה הקרחון. בספרי החדש Pro .NET Performance (הצפוי לצאת במהלך החודש הקרוב) יש כ-60 עמודים המוקדשים בלעדית לנושא איסוף הזבל (ממנו שאלתי גם את האיורים), וישנם גם מקורות נוספים באינטרנט שתוכלו להיעזר בהם להעמקה.

## על המחבר

סשה גולדשטיין הוא ה-CTO של קבוצת סלע, חברת ייעוץ, הדרכה ומיקור חוץ בינלאומית עם מטה בישראל. סשה אוהב לנבור בקרביים של Windows וה-CLR, ומתמחה בניפוי שגיאות ומערכות בעלות ביצועים גבוהים. סשה הוא מחבר הספר Pro .NET Performance, ובין היתר מלמד במכללת סלע קורסים בנושא Windows Internals ו-CLR Internals. בזמנו הפנוי, סשה כותב בלוג על נושאי פיתוח שונים.