Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“____” _____ 2023 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за першим (бакалаврським) рівнем вищої освіти
на тему
“Програмне забезпечення системи тестування жорсткого диску з використанням технології SMART”

Виконав здобувач вищої освіти
IV курсу, групи КМ-19
ОПП «Комп’ютерна інженерія»
спеціальності 123 «Комп’ютерна інженерія»
_____ Семенюк О.В.
«____» _____ 2023 р.

Керівник проекту
кандидат технічних наук
_____ Буравченко К.О.
«____» _____ 2023 р.
Рецензент _____
_____

м. Кропивницький

Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*

Кафедра *Кібербезпеки та програмного забезпечення*

Освітній ступінь *бакалавр*

Галузь знань ._12 _____*“*Інформаційні технології*”*_____

Спеціальність *123 “*Комп’ютерна інженерія*”*

Освітньо-професійна (освітньо-наукова) програма _*“*Комп’ютерна інженерія*”*_

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____ Олексій СМІРНОВ
« 17 » січня 2023 року

ЗАВДАННЯ НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ ЗА ПЕРШИМ (БАКАЛАВРСЬКИМ) РІВНЕМ ВИЩОЇ ОСВІТИ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

Семенюка Олександра Валентиновича
(прізвище, ім’я, по батькові)

1. Тема роботи *Програмне забезпечення системи тестування жорсткого диску з використанням технології SMART*

2. Керівник роботи *Буравченко Костянтин Олегович, канд. техн. наук*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 10-02 від 5.01.2023 року

3. Строк подання студентом роботи до захисту *23.05.2023 р.*

4. Мета та завдання випускної кваліфікаційної роботи*: Метою роботи є розробка програмного забезпечення системи тестування жорсткого диску з використанням технології SMART*

5. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Призначення та область використання.

2.Перегляд аналогічних існуючих систем.

3. Опис і обґрунтування проектних рішень.

4. Етапи програмування системи.

5. Впровадження системи в промислову експлуатацію.

6. Висновки

6. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

Структурна схема системи	*1 аркуш*
Функціональна схема системи	*1 аркуш*
Діаграма процесів	*1 аркуш*
Блок-схема алгоритму роботи додатку	*2 аркуша*

7. Дата видачі завдання « 17 » січня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Строк виконання етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Примітка
1.	Аналіз існуючих систем	10.03.2023 р.	
2.	Постановка задачі, оформлення ТЗ	15.03.2023 р.	
3.	Розробка моделі компонента	20.03.2023 р.	
4.	Розробка структур даних	25.03.2023 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.03.2023 р.	
6.	Програмування алгоритмів	10.04.2023 р.	
7.	Оформлення ПЗ	17.04.2023 р.	
8.	Попередній захист роботи	23.05.2023 р.	

Дата видачі завдання
« 17 » січня 2023 р.

Підпис керівника
_____ Буравченко К.О.
(прізвище та ініціали)

Завдання прийнято до виконання
« 17 » січня 2023 р.

Підпис здобувача
_____ Семенюк О.В.
(прізвище та ініціали)

АНОТАЦІЯ

Семенюк О.В. Програмне забезпечення системи тестування жорсткого диску з використанням технології SMART. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2023.

В даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи тестування жорсткого диску з використанням технології SMART.

Метою розробки є програмне забезпечення системи тестування жорсткого диску з використанням технології SMART.

Результат роботи – програмна реалізація системи тестування жорсткого диску з використанням технології SMART.

В процесі роботи над програмною моделлю виконано аналіз існуючих апаратних та програмних засобів. В повній мірі описані всі компоненти розробленого програмного забезпечення.

Розроблено зручний інтерфейс користувача. Наведені інструкції по роботі з програмними засобами.

Програма може використовуватися на ПЕОМ архітектури IBM PC з ОС Windows 10/11.

Програму розроблено в середовищі Visual C++.

Ключові слова: комп'ютерна інженерія, тестування жорсткого диску, SMART

ABSTRACT

Semeniuk O.V. Hard disk testing system software using SMART technology. 123 Computer engineering. Central Ukrainian National Technical University. Kropyvnytskyi. 2023.

In this final qualification work for the first (bachelor) level of higher education, software was developed, which is intended for the hard disk testing system using SMART technology.

The purpose of the development is the hard disk testing system software using SMART technology.

The result of the work is the software implementation of the hard disk testing system using SMART technology.

In the process of working on the software model, an analysis of existing hardware and software was performed. All components of the developed software are fully described.

A convenient user interface has been developed. Instructions for working with software tools are provided.

The program can be used on PCs of IBM PC architecture with Windows 10/11 OS.

The program was developed in the Visual C++ environment.

Keywords: computer engineering, hard disk testing, SMART

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

API	–	прикладний програмний інтерфейс
HDD	–	жорсткий диск
LBA	–	адресація секторів
MFC	–	Microsoft Foundation Class library
RO	–	тільки читання
SMART	–	Self-Monitoring, Analysis and Reporting Technology
БМГ	–	блок магнітних головок
ПЗ	–	програмне забезпечення

ВСТУП

Актуальність теми. Технологія SMART робить спостереження за основними характеристиками накопичувача, кожна з яких одержує оцінку. Характеристики можна розбити на дві групи:

– Параметри, що відбивають процес природного старіння жорсткого диска (число обертів шпинделя, число переміщень головок, кількість циклів включення-вимикання).

– Поточні параметри накопичувача (висота головок над поверхнею диска, число перепризначених секторів, час пошуку доріжки й кількість помилок пошуку).

Технологія SMART дозволяє здійснювати:

– моніторинг параметрів стану;

– сканування поверхні;

– сканування поверхні з автоматичною заміною сумнівних секторів на надійні.

Варто помітити, що технологія SMART дозволяє пророкувати вихід пристрою з ладу в результаті механічних несправностей, що становить близько 60 % причин, по яких вінчестери виходять із ладу. Пророчити наслідки стрибка напруги або ушкодження накопичувача в результаті удару SMART нездатний.

Слід зазначити, що накопичувачі не можуть самі повідомляти про свій стан за допомогою технології SMART, для цього існують спеціальні програми. Таким чином, використання технології SMART немислимо без двох складових:

– ПЗ, убудованого в контролер накопичувача.

– Зовнішнього ПЗ, убудованого в хост.

Мета й завдання дослідження. Метою роботи є програмне забезпечення системи тестування жорсткого диску з використанням технології SMART.

Для досягнення поставленої мети визначена програма дослідження, що складається з наступних завдань:

– Огляд існуючих систем тестування жорсткого диску з використанням технології SMART.

– Дослідження системи тестування жорсткого диску з використанням технології SMART.

– Програмна реалізація системи тестування жорсткого диску з використанням технології SMART.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі тестування жорсткого диску з використанням технології SMART.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи тестування жорсткого диску з використанням технології SMART, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

SMART – це технологія внутрішньої оцінки стану диска, і механізм пророкування можливого виходу з ладу жорсткого диска. Важливо відзначити те, що технологія в принципі не вирішує виникаючих проблем (основні з них показані на рисунку 1.1), вона здатна лише попередити про вже виниклу проблему або про ту, що може бути в найближчому часі.

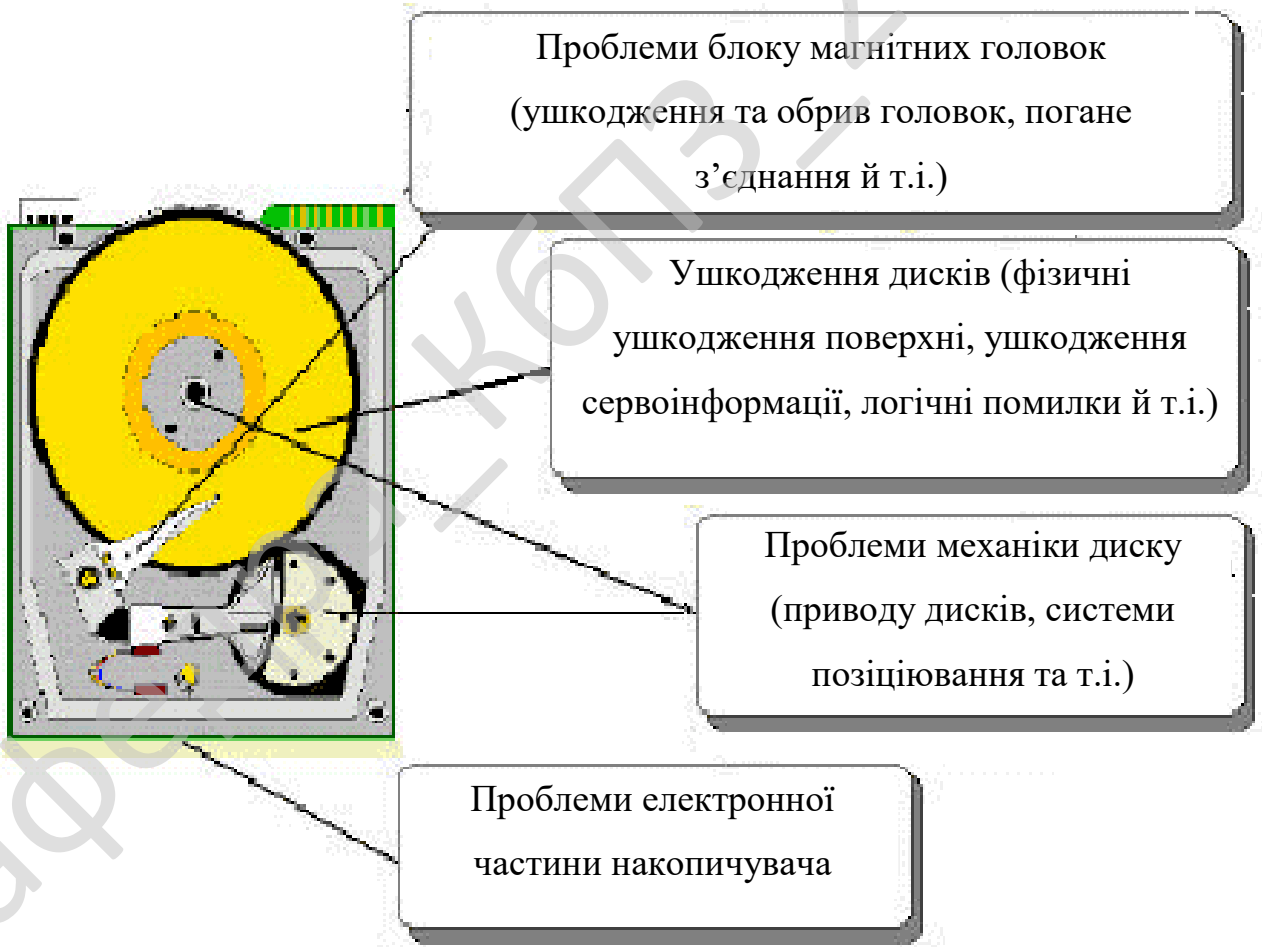

Рисунок 1.1 – Проблеми виникаючі на жорсткому диску

При цьому потрібно також сказати, що технологія не в змозі пророчити абсолютно всі можливі проблеми й це логічно: вихід електроніки в результаті стрибка напруги, псування головок і поверхні в результаті удару й т.п. ніяка технологія пророчити не в силах. Передбачувані лише ті проблеми, які пов'язані з поступовим погіршенням яких-небудь характеристик, рівномірною деградацією яких або компонент.

S.M.A.R.T. являє собою набір міні-підпрограм, які є частиною мікрокоду накопичувача й визначають підтримувані діагностичні функції. Найпоширеніші серед них:

– набір атрибутів, що відбивають стан окремих параметрів накопичувача (до 30);

– внутрішні тести накопичувача (self– test);

– журнали S.M.A.R.T. (помилок, загального стану, дефектних секторів і т.п.).

У даний момент не існує офіційної документації або стандарту на технологію S.M.A.R.T. У зв'язку із цим, виробники не публікують повні характеристики й підтримувані функції S.M.A.R.T. у своїх накопичувачах. Обов'язковий мінімум описаний в останньому стандарті ATA/ ATAPI-6.

Розвиток технології S.M.A.R.T.

Історія технології S.M.A.R.T. не так уже й багата подробицями:

– SMART I передбачав моніторинг основних життєво важливих параметрів і запускався тільки після команди по інтерфейсу.

– в SMART II з'явилася можливість фонової перевірки поверхні, що виконувалася накопичувачем автоматично під час "холостого ходу"; з'явилася функція журналювання помилок.

– в SMART III уперше з'явилася не тільки функція виявлення дефектів поверхні, але й можливість їхнього відновлення "прозоро" для користувача й багато інших нововведень.

Відомо, що першими розробили основи й запропонували цю технологію

спільно Western Digital, Seagate і Quantum. Після цього їх уже підтримали такі компанії як IBM, Maxtor і Samsung. Hitachi взяла участь у розвитку технології S.M.A.R.T. уже на стадії розробки SMART II, першими запропонувавши методику повної самодіагностики накопичувача (extended self-test).

У цей час виробники жорстких дисків готуються прийняти до використання новий варіант технології S.M.A.R.T. – "1024 S.M.A.R.T.", характерною рисою якого буде помітно більший розмір журналів, повсюдне використання мультисекторних журналів, більше точні алгоритми аналізу показань убудованих у накопичувач сенсорів (термодатчики, сенсори ударів, і т.п.) і багато чого іншого. От кілька нових функцій:

– введення алгоритму аналізу температурного режиму накопичувача;

– введення обмеження по мінімальній і максимальній температурі в робочому стані;

– введення лічильника загальної кількості записаних секторів протягом життєвого циклу накопичувача;

– введення лічильника запусків внутрішніх алгоритмів відновлення (recovery counters).

Головним же плюсом можна вважати введення нових атрибутів, які дозволять контролювати стан і робочі характеристики по кожній з головок читання/запису:

– відносна стійкість (стабільність "польоту") головки;

– виправлення помилок читання (з "схованими" повторними спробами);

– автоматичний перерозподіл дефектних ділянок поверхні при операціях запису;

– лічильник-накопичувач G-List для обліку кількості прийнятих ударних навантажень;

– лічильник-накопичувач S-List для обліку загальної кількості "програмних" помилок.

Дані зберігаються в шістнадцатковому виді, називаному «raw value», а потім перераховуються в «value», значення, що символізує надійність щодо

деякого еталонного значення. Звичайно «value» розташовується в діапазоні від 0 до 100 (деякі атрибути мають значення від 0 до 200 і від 0 до 253).

Висока оцінка говорить про відсутність змін даного параметра або повільному його погіршенні. Низька говорить про можливий швидкий збій.

Значення, менше, чим мінімальне, при якому виробником гарантується безвідмовна робота накопичувача, означає вихід вузла з ладу.

1.2 Область застосування

Програми, що відображають стан SMART-атрибутів, працюють за наступним алгоритмом:

– Перевіряють наявність підтримки технології SMART накопичувачем.

– Подають у накопичувач команду запиту SMART-таблиць.

– Одержують таблиці в буфер додатка.

– Розбирають табличні структури, витягаючи з них номера атрибутів і їхні числові значення.

– Зіставляють стандартизовані номери атрибутів їхнім назвам (іноді – залежно від типу, моделі або фірми-виробника HDD).

– Виводять числові значення в зручному для сприйняття виді.

– Витягають із таблиць прапори атрибутів (ознаки, що характеризують призначення атрибута в рамках конкретної firmware накопичувача, наприклад, «життєво важливий» або «лічильник»).

– На підставі всіх таблиць, значень і прапорів виводять загальний стан пристрою.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи тестування жорсткого диску з використанням технології SMART, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти

Сучасні жорсткі диски мають більшу кількість сучасних технологій захисту інформації. Однієї з них є «**SMART**» (Self-Monitoring, Analysis and Reporting Technology), що убудована в контролер жорсткого диска. **SMART** дивиться за станом поверхні вінчестера, виявляє, локалізує і усуває збійні області й помилки.

Цю технологію в цей час використовують всі виробники вінчестерів. Вона дозволила помістити всю технічну інформацію про збійні області усередину жорсткого диска (вони існують завжди, навіть у вінчестерах від самих надійних виробників). Тому, використовуючи спеціалізоване ПЗ, можна "вилікувати" жорсткий диск (прочитати не тільки ці необхідні дані, відновити інформацію, але й іноді виправити помилки).

Сучасний вінчестер дозволяє усунути збійні блоки шляхом заміни цієї області на резервну. У цьому випадку погана область жорсткого диска міститься в «bad-list» і більше не використовується для зберігання даних, а з резерву виділяється аналогічна по обсязі. Як правило, виробники жорстких дисків допускають заміну 100 ділянок, що, у принципі, повинне вистачити для життя вінчестера, якщо до нього не застосовувати фізичного впливу. Подібну операцію дозволяють робити спеціальні програми перевірки від виробників, однак, потрібно пам'ятати, що не кожну проблему вони можуть вирішити. Спробуємо розібратися більш ретельно.

Fujitsu FJDT

Компанія пропонує власникам IDE вінчестерів **Fujitsu** утиліту діагностики, що називається **FJDT** (Fujitsu ATA Diagnostic Tool). Сама **FJDT** займає 77 Кб і запускається із завантажувальної дискети DOS. До речі, вона працює тільки з накопичувачами **Fujitsu**. Після старту програми й вибору вінчестера, що перевіряється, користувач має можливість запустити швидкий (Quick), а потім при необхідності й розширений (Comprehensive) тести. Тривалість першого випробування – від півтора до двох мінут. У цьому випадку **FJDT** перевіряє адекватність роботи кеш-буфера при записі/читанні даних, сканує внутрішню (150 Мб) і зовнішню (50 Мб) області на предмет ушкодження поверхні, виконує операції випадкового пошуку/читання й т.п. Comprehensive Test триває довше – до 20 мінут, залежно від обсягу вінчестера. При цьому виробляється повна перевірка робочої поверхні диска. Після проходження диском випробувань програма видає резюме про придатність використання диска. При порушенні функціональності індикуються код помилки й рекомендація звернутися в службу підтримки **Fujitsu**. Якщо порушення не занадто серйозні, то програма може спробувати усунути несправність самотужки.

Maxtor Powermax

Утиліта **Powermax**, призначена для перевірки жорстких дисків цієї компанії. Вона займає 2,38 Мб. Цей програмний продукт може запускатися в середовищі Windows і OS/2, однак повністю адекватних результатів тестування в цьому випадку очікувати не доводиться. Для одержання максимально достовірної інформації **Powermax** необхідно стартувати із завантажувальної дискети. При запуску утиліти спочатку перевіряється підключення кабелю, установки перемичок master/slave, здатність системи підтримувати вінчестери великого обсягу й т.п. Після успішного проходження даного тесту (Installation Confirmation) є можливість виконати швидку (Quick), розширену (Advanced) діагностику або Burn In Test. Перший тест триває 90 сек., у ньому аналізується журнал відмов, що перебуває на спеціальній області жорсткого

диска. Також виробляється спроба читання/запису кожною головкою. Другий тест уже має на увазі перевірку поверхні вінчестера, внаслідок чого час проходження істотно збільшується. Burn In Test імітує звичайну роботу жорсткого диска, що також дозволяє виявити деякі помилки. Результатом випробувань буде повідомлення про те, що вінчестер функціонує нормально, або код помилки, якому можна використовувати для одержання більше детальних інструкцій від служби технічної підтримки.

Quantum Data Protection System

Він складається у вигляді одного exe-файлу, що служить для створення завантажувальної дискети. На ній крім операційної системи будуть утримуватися й службові файли програмного комплексу DPS. Програма від **Quantum** призначена для перевірки як SCSI, так і IDE-вінчестерів. Причому, зважаючи на те, що компанія однієї з перші початки пропонувати подібні засоби, **Data Protection System** сумісна з моделями жорстких дисків, випущених два з половиною роки тому. Інтерфейс програми гранично спрощений. Після вибору вінчестера для наступної діагностики автоматично запускається швидкий (Quick) тест, що триває півтори мінути. У процесі його виконання виробляється перевірка підключення вінчестера, тестування роботи кеш-буфера, головок читання/запису, аналіз інформації, отриманої за допомогою **SMART**, а також сканування перших 300 Мб інформації. Після цього при необхідності можна запустити повний тест робочої поверхні (Extended Test) тривалістю до 20 мінут.

IBM Drive Fitness Test

Для діагностики й усунення несправностей жорстких дисків компанія **IBM** пропонує програмне рішення за назвою **Drive Fitness Test**, що працює як з SCSI, так і з E-IDE-вінчестерами **IBM**. Крім цього, програма існує як для Windows, так і для Linux. **Drive Fitness Test** складається з мікрокоду, що зберігається в спеціальних областях жорсткого диска й що дозволяє вести облік його відмов, а також програмної частини, що запускається із системної дискети. Утиліта **Drive Fitness Test** складається з наступних розділів: Utilities, Fitness Test

і Help. У секції Utilities є можливість включити або відключити підтримку **SMART** жорсткого диска, обнулити boot-сектор або повністю очистити накопичувач, а також одержати інформацію про підключені до системи HDD (наводяться дані про модель, обсяг диска, що тече режимі Ultra ATA, і деяких установках, наприклад, Write Cashe, Read Look Ahead і т.п.). У розділі Fitness Test можна запускати три типи тестів: Quick, Advanced (Media Scan) і Exerciser. Швидкий тест триває менш двох мінут. При цьому, в основному, аналізується журнал відмов, що зберігається на самому жорсткому диску, виробляється читання й запис кожною головкою, а також скануються перші півмегабайта поверхні. Незважаючи на те, що даний тест нетривалий, він дозволяє виявити й класифікувати до 90% відмов жорсткого диска. У режимі Media Scan крім швидкого тесту виробляється також перевірка поверхні жорсткого диска, що дає можливість визначити проблеми, пов'язані з механічними ушкодженнями. Саме випробування триває від 15 до 30 мінут, залежно від ємності вінчестера. Exerciser має на увазі імітацію роботи жорсткого диска зі звичайним навантаженням шляхом послідовного читання/запису даних і аналізу працездатності вінчестера в такому режимі. При цьому тривалість цього тесту задається користувачем. При виявленні помилок у функціонуванні диска в якому-небудь із перерахованих тестів утиліта **Drive Fitness Test** інформує користувача про можливість виправлення зазначених неполадок засобами самої утиліти або необхідності звернутися в сервісний центр. Крім того, на цій же сторінці можна знайти утиліту **Ontrack Data Advisor**. Вона служить для визначення проблем доступу до даних, аналізує файлову систему, сектори завантаження, MBR, пам'ять і т.д.

Seagate Disk Diagnostic

Утиліта **Disk Diagnostic** із состава комплексу **SeaTools** має самий цікавий інтерфейс (незважаючи на те, що програма запускається під керуванням DOS, тут є вкраплення графіки) і містить у собі найбільша кількість додаткових можливостей. Програма дозволяє провести два режими діагностики: швидкий (Short) і розширений (Extended), алгоритми яких такі ж, як і в інших розглянутих

утиліт. Однак, на відміну від інших програм, в **Disk Diagnostic** є можливість як запустити тести для вінчестерів **Seagate** (з урахуванням внутрішніх відмінностей цих дисків), так і провести аналогічну діагностику для всіх інших типів HDD. Крім того, що з основного меню програми можна переглянути файл readme з досить докладним її описом, кожне розкривається окно, що, супроводжується детальним текстовим описом поточного режиму й змісту проведених у даний момент тестів. Результати всіх випробувань зберігаються на дискеті в текстовому файлі й доступні для перегляду із самої утиліти. У випадку якщо не вдається відновити працездатність вінчестера засобами **Disk Diagnostic**, є можливість сформувати стандартний бланк повернення накопичувача, що буде збережений у текстовому файлі на дискеті.

Samsung SHDIAG

Утиліту діагностики **SHDIAG** ми знайшли на українському сайті компанії **Samsung Electronics**. У плані інтерфейсу й функціональності програма гранично проста. Після вибору вінчестера для діагностики виробляється швидке (тривалістю 90 секунд) тестування важливих вузлів вінчестера (кеш-буфер, головки читання/запису й т.п.), а потім, якщо знадобиться, повний тест поверхні. На відміну від інших програм, які у випадку помилки повідомляють користувача її код, **SHDIAG** просто інформує про те, що «деяка проблема має місце», і пропонує зробити низькорівневе форматування диска. Також на сторінці є програма діагностики, програма низькорівневого форматування жорсткого диска й ще кілька корисних «примочок».

Western Digital Data Lifeguard

Компанія поширює комплекс програм за назвою **Data Lifeguard**. У його состав входять наступні продукти: EZ-**Install**, **Diagnostics**, **BIOS Check** і **Ultra ATA Manager**. Запуск **Data Lifeguard** відбувається із завантажувальної дискети, на якій, крім властиво утиліт, що мають загальну оболонку, перебуває DOS-подібна операційна система. EZ-Install дозволяє швидко розбити новий вінчестер на розділи, відформатувати їх, а також скопіювати на новий HDD цілі розділи з іншого вінчестера «один-в-один». При її установці на жорсткий диск в boot-

сектор може записуватися службова інформація. Це дає можливість використовувати повний робочий обсяг вінчестера в системах, BIOS яких має обмеження по цьому параметрі. У цьому випадку розпізнавання системою вінчестера відбувається «в обхід» BIOS, і зміни настроювань жорсткого диска в BIOS Setup не будуть мати ніякого ефекту. Програма **Data Lifeguard Diagnostics** призначена для виявлення й виправлення помилок, що приводять до втрати даних. Тут доступні два режими: **Quick Test** і **Enhanced Test**. Перший служить для експрес-перевірки цілісності даних на диску на основі попередніх сканувань у фоновому режимі. Сама процедура триває півтори мінути, після чого видаються звіт про результати й рекомендацію із проведення більше ретельної перевірки у випадку виявлення помилок. Тривалість розширеного тесту становить від 10 до 40 мінут. У цьому режимі безпосередньо перевіряється вся робоча область вінчестера, після чого утиліта інформує користувача про відсутність помилок, необхідності звернутися в сервісний центр або можливості відновлення збійних ділянок засобами **Data Lifeguard** (Repair Drive). Найбільш потужним засобом діагностики служить деструктивний тест Write Zero To Drive. На відміну від розширеного тесту, що виконує неруйнуюче тестування й у випадку рекартування несправної ділянки, коли втрата інформації неминуча, сповіщає про це користувачеві, виконання Write Zero приводить до повної втрати інформації на поверхні жорсткого диска. За принципом дії цей тест дуже нагадує ніколи існуючу в системної BIOS можливість низькорівневого форматування за допомогою опції **HDD Low Level Format**. В обох випадках робота з накопичувачем ведеться через порти. Відомо досить багато ситуацій, коли жорсткий диск не визначався в системної BIOS і відновлював свою працездатність після «псевдонизькорівневого форматування» утилітою DLG. Методика полягає в тому, що при таких несправностях в Standard CMOS Setup варто вказати відсутність всіх IDE/ ATAPI-пристроїв. Якщо підключені компоненти апаратно справні, утиліта DLG виявить їх, а тест не тільки перевірить всі ділянки на запис/читання, але й видалить наявну інформацію, у тому числі й ту, котра приводила до відмови накопичувача на логічному рівні.

Утиліта BIOS Check використовується для одержання інформації про системний BIOS, а також застосовуваних HDD, що може знадобитися при зверненні до служби технічної підтримки, а за допомогою Ultra ATA Management можна примусово встановити необхідний режим Ultra ATA вінчестера.

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Для реалізації програми мною була використана мова програмування Visual C++. У зв'язку з тим, що сьогодні рівень складності програмного забезпечення дуже високий, розробка застосунків Windows з використанням тільки якої-небудь мови програмування значно утрудняється. Програміст повинен затратити масу часу на рішення стандартних завдань по створенню багатовіконного інтерфейсу. Реалізація технології зв'язування й вбудовування об'єктів – OLE – зажадає від програміста ще більш складної роботи. Щоб полегшити роботу програміста практично всі сучасні компілятори з мови C++ містять спеціальні бібліотеки класів. Такі бібліотеки містять у собі практично весь програмний інтерфейс Windows і дозволяють користуватися при програмуванні засобами більш високого рівня, чим звичайні виклики функцій. За рахунок цього значно спрощується розробка застосунків, що мають складний інтерфейс користувача, полегшується підтримка технології OLE і взаємодія з базами даних. Сучасні інтегровані засоби розробки застосунків Windows дозволяють автоматизувати процес створення застосунка. Для цього використовуються генератори застосунків. Програміст відповідає на питання генератора застосунків і визначає властивості застосунка – чи підтримує воно багатовіконний режим, технологію OLE, тривимірні органи керування, довідкову систему. Генератор застосунків, створить додаток, що відповідає вимогам, і надасть вихідні тексти. Користуючись їм як шаблоном, програміст зможе швидко розробляти свої застосунки. Подібні засоби автоматизованого створення застосунків включені в компілятор Microsoft Visual C++ і називаються MFC

AppWizard. Заповнивши кілька діалогових панелей, можна вказати характеристики застосунка й одержати його тексти, постачені великими коментарями. MFC AppWizard дозволяє створювати одновіконні й багатовіконні застосунки, а також застосунки, що не мають головного вікна, – замість нього використовується діалогова панель. Можна також включити підтримку технології OLE, баз даних, довідкової системи. Звичайно, MFC AppWizard не всесильний. Прикладну частину застосунка програмістові прийдеться розробляти самостійно. Вихідний текст застосунка, створений MFC AppWizard, стане тільки основою, до якої потрібно підключити інше. Але працюючий шаблон застосунка – це вже половина всієї роботи. Вихідні тексти застосунків, автоматично отриманих від MFC AppWizard, можуть становити сотні рядків тексту. Набір його вручну був би дуже стомлюючий. Потрібно відзначити, що MFC AppWizard створює тексти застосунків тільки з використанням бібліотеки класів MFC (Microsoft Foundation Class library). Тому тільки вивчивши мову C++ і бібліотеку MFC, можна користуватися засобами автоматизованої розробки й створювати свої застосунки в найкоротший термін. Як уже згадувався, MFC – це базовий набір (бібліотека) класів, написаних мовою C++ і призначених для спрощення й прискорення процесу програмування для Windows. Бібліотека містить багаторівневу ієрархію класів, що нараховує близько 200 членів. Вони дають можливість створювати Windows-застосунки на базі об’єктно-орієнтованого підходу. З погляду програміста, MFC являє собою каркас, на основі якого можна писати програми для Windows. Бібліотека MFC розроблялася для спрощення завдань, що стоять перед програмістом. Як відомо, традиційний метод програмування під Windows вимагає написання досить довгих і складних програм, що мають ряд специфічних особливостей. Зокрема, для створення тільки каркаса програми таким методом знадобиться близько 75 рядків коду. У міру ж збільшення складності програми її код може досягати воістину неймовірних розмірів. Однак та ж сама програма, написана з використанням MFC, буде приблизно в три рази менше, оскільки більшість приватних деталей приховано від програміста.

Одною з основних переваг роботи з MFC є можливість багаторазового використання того самого коду. В зв'язку з тим, що бібліотека містить багато елементів, загальних для всіх Windows-застосунків, немає необхідності щораз писати їх заново. Замість цього їх можна просто успадковувати (говорячи мовою об'єктно-орієнтованного програмування). Крім того, інтерфейс, забезпечуваний бібліотекою, практично незалежний від конкретних деталей, його що реалізують. Тому програми, написані на основі MFC, можуть бути легко адаптовані до нових версій Windows (на відміну від більшості програм, написаних звичайними методами). Ще однією істотною перевагою MFC є спрощення взаємодії із прикладним програмним інтерфейсом (API) Windows. Будь-який додаток взаємодіє з Windows через API, що містить кілька сотень функцій. Значний розмір API утрудняє спроби зрозуміти й вивчити його цілком. Найчастіше навіть складно простежити, як окремі частини API зв'язані один з одним! Але оскільки бібліотека MFC поєднує (шляхом інкапсуляції) функції API у логічно організовану безліч класів, інтерфейсом стає значно легше управляти.

Оскільки MFC являє собою набір класів, написаних мовою C++, тому програми, написані з використанням MFC, повинна бути в той же час програмами на C++. Для цього необхідно володіти відповідними знаннями. Для початку необхідно вміти створювати власні класи, розуміти принципи спадкування й уміти перевизначати віртуальні функції. Хоча програми, що використовують бібліотеку MFC, звичайно не містять занадто специфічних елементів з арсеналу C++, для їхнього написання проте потрібні солідні знання в даній області.

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, реалізації підлягає програмне забезпечення, яке призначено для системи тестування жорсткого диску з використанням технології SMART.

В процесі розробки випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти необхідно виконати наступний обсяг роботи:

а) провести аналіз існуючих систем-аналогів для виявлення їх позитивних і негативних якостей. Результати аналізу врахувати в подальших розробках;

б) вибрати та обґрунтувати методику побудови системи контролю роботи технологічного обладнання на виробництві в автоматизованому режимі. Розробити функціональну та структурну схеми системи;

в) розробити програмне забезпечення системи, що дозволить реалізувати поставлену технічним завданням задачу. Побудувати блок-схеми алгоритмів програми та підпрограми;

г) організувати інтерфейс користувача з метою формування та виводу на екран ЕОМ повідомлень про некоректні дії користувача та нестандартні ситуації в роботі технологічного обладнання;

д) розробити рекомендації по організаційних та методичних заходах, які забезпечать впровадження системи в промислову експлуатацію та її подальшу успішну експлуатацію;

е) провести розрахунки по визначенню економічної ефективності розробленої системи;

ж) розробити заходи по охороні праці при впровадженні та експлуатації системи, а також розробити заходи з цивільного захисту;

з) сформувати висновки про виконаний обсяг робіт та одержані результати.

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Опис технології S.M.A.R.T.

Технологія S.M.A.R.T. – Self-Monitoring, Analysis and Reporting Technology (від англ. "Технологія Самодіагностики, Аналізу й Звіту") – була розроблена для підвищення надійності й схоронності даних на жорстких дисках. У більшості випадків, SMART-сумісні пристрої дозволяють пророчити появу найбільш імовірних помилок і, тим самим, дають користувачеві можливість вчасно зробити резервну копію даних і/або повністю замінити накопичувач до виходу його з ладу.

Атрибути

Атрибути S.M.A.R.T. – особливі характеристики, які використовуються при аналізі стану й запасу продуктивності накопичувача. Атрибути вибираються виробником накопичувача, ґрунтуючись на здатності цих атрибутів пророкувати погіршення робочих характеристик накопичувача або визначити його дефектність. Кожний виробник має свій характерний набір атрибутів і може вільно вносити зміни в цей набір у відповідності зі своїми власними вимогами й без повідомлення про це фірм-продавців і кінцевих користувачів.

Значення атрибутів

Значення атрибутів (value) використовуються для подання відносної надійності окремого експлуатаційного або еталонного атрибута. Припустиме значення атрибута лежить у діапазоні від 1 до 255. Високе значення атрибута говорить про те, що результат аналізу даної робочої характеристики вказує на низьку ймовірність її погіршення або виходу накопичувача з ладу. Відповідно, низьке значення атрибута говорить про те, що результат аналізу даної робочої

характеристики вказує на високу ймовірність її погіршення або виходу накопичувача з ладу.

Граничні значення атрибутів

Кожний атрибут має власне граничне значення (threshold), що використовується для порівняння зі значенням атрибута (value) і вказує на погіршення робочих характеристик або дефектність накопичувача. Числове значення граничного атрибута визначається виробником накопичувача через конструкційні особливості накопичувача й аналіз результатів випробувань на надійність. Граничне значення кожного атрибута вказує на нижню припустиму границю значення атрибута, аж до якої зберігається позитивний статус надійності.

Граничні значення встановлюються в заводських умовах виробником накопичувача й, у більшості випадків, можуть бути змінені тільки після перемикання накопичувача в технологічний (factory mode). Припустиме граничне значення атрибута може перебуває в діапазоні від 1 до 255.

Якщо значення одного або більше атрибутів, що мають тип pre-failure (в HDD Speed відзначаються символом "*"), менше або дорівнює відповідного граничного значення, то це свідчить про майбутнє погіршення робочих характеристик і/або повному виході накопичувача з ладу.

Короткий опис основних атрибутів

Даний перелік атрибутів є найбільш повним з доступних на сьогоднішній момент у інтернеті або інших джерелах. Призначення атрибутів і спосіб інтерпретації їхніх значень виявлені або досвідченим шляхом, або отримані від служб технічної підтримки компаній-виробників накопичувачів.

Нижче наведена зведена таблиця всіх відомих атрибутів (55) і короткий опис до більшості (38) з них.

Таблиця 3.1 – Значення атрибутів

ID	Назва атрибута
0	= атрибут не використовується
1	Raw Read Error Rate
2	Throughput Performance
3	Spin Up Time
4	Start/Stop Count
5	Reallocated Sector Count
6	Read Channel Margin
7	Seek Error Rate
8	Seek Time Performance
9	Power-On Hours Count
10	Spin Retry Count
11	Recalibration Retries
12	Device Power Cycle Count
13	Soft Read Error Rate
??	Emergency Re-track (Hitachi)
??	ECC On-The-Fly Count (Hitachi)
96	? (Maxtor)
97	? (Maxtor)
98	? (Maxtor)
99	? (Maxtor)
100	? (Maxtor)
101	? (Maxtor)
191	G-Sense Error Rate
192	Power-Off Retract Cycle
193	Load/Unload Cycle Count
194	Temperature
195	? (Quantum AS, Seagate, Maxtor)
196	Reallocation Events Count

Продовження таблиці 3.1

ID	Назва атрибута
197	Current Pending Sector Count
198	Uncorrectable Sector Count
199	UltraDMA CRC Error Rate
200	Write Error Rate (в WD – MultiZone Error Rate)
201	TA Counter Detected
202	TA Counter Increased
203	? (Maxtor)
204	? (Maxtor)
205	? (Maxtor)
206	? (Maxtor)
207	? (Maxtor)
208	? (Maxtor)
209	? (Maxtor)
220	Disk Shift
221	G-Sense Error Rate (в Hitachi – Shock Sense Error Rate)
222	Loaded Hours
223	Load/Unload Retry Count
224	Load Friction
225	Load/Unload Cycle Count
226	Load-in Time
227	Torque Amplification Count
228	Power-Off Retract Count
229	? (IBM DTTA, thanx to Vladislav Shaklein)
230	GMR Head Amplitude
231	Temperature
240	Head Flying Hours (Hitachi)
250	Read Error Retry Rate

Короткий опис відомих атрибутів:

– * – (використовується в програмі HDD Speed) – даний покажчик показує, що відповідний атрибут S.M.A.R.T. є критичним для нормального функціонування накопичувача. Погіршення значень таких атрибутів з найбільшою ймовірністю приводить до виходу накопичувача з ладу. У нових материнських платах BIOS мають убудовану функцію контролю стану накопичувача саме по цих атрибутах.

– Raw Read Error Rate – Частота появи помилок при читанні даних з диска. – Даний параметр показує частоту появи помилок при операціях читання з поверхні диска з вини апаратної частини накопичувача.

– Throughput Performance – Середня продуктивність (пропускна здатність) диска. – Зменшення значення value цього атрибута з великою ймовірністю вказує на проблеми в накопичувачі.

– Spin Up Time – Час розкручування шпинделя. – Середній час розкручування шпинделя диска від 0 RPM до робочої швидкості. Можливо, у поле raw value утримується час у міллісекундах/секундах.

– Start/Stop Count – Кількість циклів запуск/останов шпинделя. – Поле raw value зберігає загальна кількість включень/вимикань диска.

– Reallocated Sectors Count – Кількість перепризначених секторів. – Коли жорсткий диск зустрічає помилку читання/запису/верифікації він намагається перемістити дані з нього в спеціальну резервну область (spare area) і, у випадку успіху, позначає сектор як "перепризначений". Також, цей процес називають remapping, а перепризначений сектор – remap. Завдяки цій можливості, на сучасних жорстких дисках дуже рідко видні [при тестуванні поверхні] так звані bad block. Однак, при великій кількості ремапів, на графіку читання з поверхні будуть помітні "провали" – різке падіння швидкості читання (до 10% і більше). Поле raw value містить загальна кількість перепризначених секторів.

– Read Channel Margin – Запас каналу читання. – Призначення цього атрибута не документовано й у сучасних накопичувачах він не використовується.

– Seek Error Rate – Частота появи помилок позиціонування блоку магнітних головок (БМГ). – У випадку збоїти в механічній системі позиціонування, ушкодження сервометок (servo), сильного термічного розширення дисків і т.п. виникають помилки позиціонування. Чим їх більше, тим гірше стан механіки й/або поверхні жорсткого диска.

– Seek Time Performance – Середня продуктивність операцій позиціонування БМГ. – Даний параметр показує середню швидкість позиціонування привода БМГ на зазначений сектор. Зниження значення цього атрибута говорить про неполадки в механіку привода.

– Power-On Hours – Кількість відпрацьованих годин у включеному стані. – Поле raw value цього атрибута показує кількість годин (мінут, секунд – залежно від виробника), відпрацьованих жорстким диском. Зниження значення (value) атрибута до критичного рівня (threshold) указує на виробіток диском ресурсу (MTBF – Mean Time Between Failures). На практиці, навіть падіння цього атрибута до нульового значення не завжди вказує на реальне вичерпування ресурсу й накопичувач може продовжувати нормально функціонувати.

– Spin Retry Count – Кількість повторів спроб старту шпинделя диска. – Даний атрибут фіксує загальну кількість спроб розкручування шпинделя і його виходу на робочу швидкість, за умови, що перша спроба була невдалою. Зниження значення цього атрибута говорить про неполадки в механіку привода.

– Recalibration Retries – Кількість повторів спроб рекалібровки накопичувача. – Даний атрибут фіксує загальну кількість спроб скидання стану накопичувача й установки головок на нульову доріжку, за умови, що перша спроба була невдалою. Зниження значення цього атрибута говорить про неполадки в механіку привода.

– Device Power Cycle Count – Кількість повних циклів запуску/останова жорсткого диска.

– Soft Read Error Rate – Частота появи "програмних" помилок при читанні даних з диска. – Даний параметр показує частоту появи помилок при операціях

читання з поверхні диска з вини програмного забезпечення, а не апаратної частини накопичувача.

– Emergency Re-track

– ECC On-The-Fly Count

– Load/Unload Cycle Count – Кількість циклів виводу БМГ у спеціальну парковочну зону/у робоче положення. Докладніше – опис технології Head Load/Unload Technology.

– Temperature – Температура. – Даний параметр відбиває в поле raw value показання убудованого температурного сенсора в градусах Цельсія.

– Reallocation Event Count – Кількість операцій перепризначення (ремаппінгу). – Поле raw value цього атрибута показує загальну кількість спроб перепризначення збійних секторів у резервну область, початих накопичувачем. При цьому, ураховуються як успішні, так і невдалі операції.

– Current Pending Sector Count – Поточна кількість нестабільних секторів. – Поле raw value цього атрибута показує загальну кількість секторів, які накопичувач у цей момент уважає претендентами на перепризначення в резервну область (remap). Якщо надалі якийсь із цих секторів буде прочитаний успішно, то він виключається зі списку претендентів. Якщо ж читання сектора буде супроводжуватися помилками, то накопичувач спробує відновити дані й перенести їх у резервну область, а сам сектор позначити як перепризначений (remapped). Постійно ненульове значення raw value цього атрибута говорить про низьку якість (окремої зони) поверхні диска.

– Uncorrectable Sector Count – Кількість нескоректованих помилок. – Атрибут показує загальну кількість помилок, що виникли при читанні/запису сектора і які не вдалося скорегувати. Ріст значення в поле raw value цього атрибута вказує на явні дефекти поверхні й/або проблеми в роботі механіки накопичувача.

– UltraDMA CRC Error Count – Загальна кількість помилок CRC у режимі UltraDMA. – Поле raw value містить кількість помилок, що виникли в режимі

передачі даних UltraDMA у контрольній сумі (ICRC – Interface CRC). Практика, зібрана статистика й вивчення журналів помилок SMART показують: у більшості випадків помилки CRC виникають при сильному завищенні частоти PCI (більше номінальних 33.6 MHz), сильно перекрученому кабелі, а також – з вини драйверів ОС, які не дотримують вимог до передачі/прийому даних у режимах UltraDMA.

– Write Error Rate (Multi Zone Error Rate) – Частота появи помилок при записі даних. – Показує загальну кількість помилок, виявлених під час запису сектора. Чим більше значення в поле raw value (і нижче значення value), тим гірше стан поверхні диска й/або механіки привода.

– Disk Shift – Зрушення пакета дисків щодо осі шпинделя. – Актуальне значення атрибута втримується в поле raw value. Одиниці виміру – не відомі. Подробиці в описі технології G-Force Protection. Зрушення пакета дисків можливий у результаті сильного ударного навантаження на накопичувач у результаті його падіння або з інших причин.

– G-Sense Error Rate – Частота появи помилок у результаті ударних навантажень. – Даний атрибут зберігає показання ударочуттєвого сенсора – загальна кількість помилок, що виникли в результаті отриманих накопичувачем зовнішніх ударних навантажень (при падінні, неправильній установці, і т.п.). Докладніше в описі технології G-Force Protection.

– Loaded Hours – Навантаження на привод БМГ, викликана загальним наробітком годин накопичувачем. – Ураховується тільки період, у плині якого головки перебували в робочому положенні.

– Load/Unload Retry Count – Навантаження на привод БМГ, викликана численними повтореннями операцій читання, запису, позиціонування головок і т.п. Ураховується тільки період, у плині якого головки перебували в робочому положенні.

– Load Friction – Навантаження на привод БМГ, викликане тертям у механічних частинах накопичувача. – Ураховується тільки період, у плині якого головки перебували в робочому положенні.

– Load/Unload Cycle Count – Загальна кількість циклів навантаження на привод БМГ. – Ураховується тільки період, у плині якого головки перебували в робочому положенні.

– Load-in Time – Загальний час навантаження на привод БМГ. – Можливо, даний атрибут показує загальний час роботи накопичувача під навантаженням, за умови, що головки перебувають у робочому стані (поза парковочною зоною).

– Torque Amplification Count – Кількість зусиль обертаючого моменту привода.

– Power-Off Retract Count – Кількість зафіксованих повторів включення/вимикання живлення накопичувача.

– GMR Head Amplitude – Амплітуда тремтіння головок (GMR-head) у робочому стані.

– Head Flying Hours

– Read Error Retry Rate

Типи атрибутів

Кожний атрибут може мати деякий набір прапорів, що визначають його функціональні особливості. Нижче приводяться всі шість основних типів і їхні короткі описи:

– Pre-failure (PF). Якщо атрибут має цей тип, то поле threshold атрибута містить мінімально припустиме значення атрибута, нижче якого не гарантується працездатність накопичувача й різко збільшується ймовірність його виходу з ладу.

– On-line collection (OC). Указує, що значення даного атрибута обновляється (обчислюється) під час виконання on-line тестів S.M.A.R.T. або ж під час обох видів тестів (on-line/off-line). У противному випадку, значення атрибута обновляється тільки при виконанні off-line тестів.

– Performance related (PR). Указує на те, що значення цього атрибута прямо залежить від продуктивності накопичувача за окремими показниками

(seek/throughput/etc. performance). Звичайно обновляється після виконання self-test'ов SMART.

– Error rate (ER). Указує на те, що значення атрибута відбиває відносну частоту помилок по даному параметрі (raw read/write, seek, etc.).

– Events count (EC). Указує на те, що атрибут є лічильником подій.

– Self-preserve (SP). Указує на те, що значення атрибута обновляється й зберігається автоматично (звичайно при кожному старті накопичувача й при виконанні тестів SMART).

Автономне сканування поверхні (off-line read scanning)

Більшість накопичувачів забезпечують підтримку автономного сканування поверхні, що є однієї з функцій підпрограми автономного збору даних про стан накопичувача (off-line data collection). При виконанні цієї функції, накопичувач виконує повне сканування поверхні шляхом читання кожного сектора й заміщенням ненадійних секторів на запасні сектори з резервної області (spare area) для запобігання втрати користувальницьких даних.

Якщо під час виконання сканування накопичувач одержує команду по інтерфейсу, то процес сканування переривається й накопичувач приступає до обробки команди, що надійшла. При цьому гарантується максимальний час реагування на команду, що надійшла, – до 2 секунд.

Журнали помилок (SMART error log)

У більшості сучасних накопичувачів реалізована функція журналювання, помилок або інших подій, що з'являються в плині роботи накопичувача. В основному, накопичувачі надають інформацію про п’ять останіх помилок. При цьому зберігаються останні 5 команд, що надійшли в накопичувач, що передують виникненню цієї помилки, і інша необхідна інформація. Накопичувач може також підтримувати додаткові журнали. Їхня структура, розмір і призначення встановлюються фірмою-виробником. При відновленні мікропрограми накопичувача, всі журнали накопичувача очищаються, а загальна кількість помилок установлюється в значення 0.

У журналах зберігається час по внутрішніх годинниках накопичувача, тобто або загальний відпрацьований час на даний момент, або час від моменту останнього включення накопичувача.

Log Directory

Тип: Каталог журналів S.M.A.R.T.

Вид доступу: тільки читання (RO).

Розмір: 1 сектор (512 байт).

Примітка: підтримка мультисекторних журналів.

Даний журнал являє собою свого роду каталог, у якому зазначені адреси всіх підтримуваних журналів S.M.A.R.T. і їхній розмір у секторах. Максимальна кількість журналів – 255.

Summary Error Log

Тип: Сумарний журнал помилок.

Вид доступу: тільки читання (RO).

Розмір: 1 сектор (512 байт).

Примітка: підтримується тільки 28-бітна адресація секторів (28-bit LBA).

Даний журнал містить інформацію про загальну кількість помилок, зафіксованих накопичувачем з моменту першого включення (або відновлення мікропрограми) і докладні записи про останні 5 помилок. Для кожної з 5 зафіксованих помилок зберігаються останні 5 команд, що надійшли в накопичувач. У цьому журналі зберігаються всі помилки UNC, IDNF, помилки сервосистеми, запису/читання й т.д. При цьому, для кожної команди зберігається значення всіх регістрів, час і поточний стан накопичувача на момент подачі самої команди. Помилки, викликані подачею непідтримуваних команд або командами з помилковими параметрами не фіксуються в журналі. Якщо накопичувач підтримує Comprehensive Error Log, то журнал Summary Error Log дублює останні п'ять записів з журналу Comprehensive Error Log.

Comprehensive Error Log

Тип: Комплексний журнал помилок [SMART Error Logging].

Вид доступу: тільки читання (RO).

Розмір: 1..51 сектор (максимум 26,112 байт).

Примітка: підтримується тільки 28-бітна адресація секторів (28-bit LBA).

Даний журнал містить докладну інформацію про загальну кількість помилок, зафіксованих накопичувачем з моменту першого включення (або відновлення мікропрограми) і докладні записи про останні помилки. Максимальна кількість помилок, що зберігаються – 255. Для кожної зафіксованої помилки зберігаються останні 5 команд, що надійшли в накопичувач. У цьому журналі зберігаються всі помилки UNC, IDNF, помилки сервосистеми, запису/читання й т.д. При цьому, для кожної команди зберігається значення всіх регістрів, час і поточний стан накопичувача на момент подачі самої команди. Помилки, викликані подачею непідтримуваних команд або командами з помилковими параметрами не фіксуються в журналі.

Extended Comprehensive Error Log

Тип: Розширений комплексний журнал помилок [SMART Error Logging].

Вид доступу: тільки читання (RO).

Розмір: 1..65,536 секторів (максимум 32 Мбайт).

Примітка: підтримується 28/ 48-бітна адресація секторів.

Призначення даного журналу аналогічно журналу Comprehensive Error Log і містить у собі копію його записів, однак цей журнал має іншу структуру, що дозволяє реалізувати підтримку як 28-бітної, так і 48-бітної адресації секторів. Максимальна кількість помилок, що зберігаються – 327,680.

Self-test Log

Тип: Журнал результатів самоконтролю [SMART self-test].

Вид доступу: тільки читання (RO).

Розмір: 1 сектор (512 байт).

Примітка: підтримується тільки 28-бітна адресація секторів (28-bit LBA).

Даний журнал містить інформацію про результати виконання команд внутрішньої самодіагностики накопичувача. Журнал може зберігати до 21

запису. При перевищенні цієї кількості, журнал починає заповнюватися заново, перезаписуючи 1-й запис 22-м, 2-й – 23-м і так далі. У кожному записі журналу зберігається регістр із номером тесту, код статусу виконання тесту, час на момент запуску/переривання тесту, номер поточної контрольної точки (або точки останова) тесту, а також LBA-адресу сектора, на якому відбулося переривання/скасування тесту.

Extended Self-test Log

Тип: Розширений журнал результатів самоконтролю [SMART self-test].

Вид доступу: тільки читання (RO).

Розмір: 1..65,536 секторів (максимум 32 Мбайт).

Примітка: підтримується 28/ 48-бітна адресація секторів.

Призначення даного журналу аналогічно журналу Self-test Log і містить у собі копію його записів, однак цей журнал має іншу структуру, що дозволяє реалізувати підтримку як 28-бітної, так і 48-бітної адресації секторів. Максимальна кількість записів – 1,179,648.

Streaming Performance Log

Тип: Журнал параметрів продуктивності потоків [Streaming].

Вид доступу: тільки читання (RO).

Розмір: 1..65,536 секторів (максимум 32 Мбайт).

Даний журнал містить інформацію про переданий накопичувачу параметрів командами керування режимом Automatic Acoustic Management і Typical Host Interface Sector Time (докладніше – див. ATA/ ATAPI-6 rev 1e). У журналі зберігається набір параметрів, по яких виробляється настроювання накопичувача й переклад у нього в режим, коли всі операції читання/запису можливі тільки спеціальними командами й передача даних відбувається у вигляді безперервного потоку, для якого гарантовані й ураховуються всі часові інтервали (на обробку команди, читання й передачу даних; мінімальні/максимальні затримки, час доступу, позиціонування й т.п.). Докладніше про призначення

даного виду журналів можна довідатися з опису технології Audio/Video (AV) Streaming Feature.

Write Stream Error Log

Тип: Журнал помилок потокового запису [Streaming].

Вид доступу: тільки читання (RO).

Розмір: 1 сектор (512 байт).

Примітка: підтримується 48-бітна адресація секторів.

Даний журнал містить інформацію про виниклі помилки запису в період роботи накопичувача в потоковому режимі (streaming mode). У цьому журналі зберігається загальна кількість подібних помилок, номер останньої помилки, попереднє і поточне значення регістрів стану й помилки, кількість і LBA-номер сектора, на якому дана помилка була зафіксована. Після читання даного журналу, накопичувач скидає лічильник загальної кількості помилок і очищає журнал. Вміст журналу зберігається тільки під час роботи й очищається в момент наступного включення/вимикання накопичувача або при надходженні сигналу апаратного скидання (hardware reset). Максимальна кількість помилок, що зберігаються – 31.

Read Stream Error Log

Тип: Журнал помилок потокового читання [Streaming].

Вид доступу: тільки читання (RO).

Розмір: 1 сектор (512 байт).

Примітка: підтримується 48-бітна адресація секторів.

Даний журнал містить інформацію про виниклі помилки читання в період роботи накопичувача в потоковому режимі (streaming mode). У цьому журналі зберігається загальна кількість подібних помилок, номер останньої помилки, попереднє і поточне значення регістрів стану й помилки; кількість і LBA-номер сектора, на якому дана помилка була зафіксована. Після читання даного журналу, накопичувач скидає лічильник загальної кількості помилок і очищає журнал. Вміст журналу зберігається тільки під час роботи й очищається в момент

наступного включення/вимикання накопичувача або при надходженні сигналу апаратного скидання (hardware reset). Максимальна кількість помилок, що зберігаються – 31.

Delayed LBA Sector Log

Тип: Vendor Specified [General Purpose Logging].

Вид доступу: тільки читання (RO).

Розмір: установлюється виробником (VS).

Примітка: підтримується 48-бітна адресація секторів.

Даний журнал містить LBA-адреси всіх секторів, які були переміщені зі свого нормального фізичного розташування, а також адреси границь недоступної послідовності секторів. У такий спосіб ведеться журнал всіх дефектних або нестабільних секторів. Максимальний розмір журналу встановлюється виробником. Нове фізичне розташування, метод і час доступу до заміщених секторів також установлюється виробником і не документується. Запис у даний журнал може бути додана в будь-який момент часу, за умови активності (живлення) самого накопичувача. Для процесу відновлення журналу встановлюється найвищий пріоритет і виконання всіх інших команд припиняється. При цьому видалити існуючий запис із журналу не можливо. Вміст журналу зберігається при циклах включення/вимикання накопичувача й при надходженні сигналу апаратного скидання (hardware reset).

ECC Uncorrectable Sector Log

Тип: Журнал непоправних помилок ECC [SMART Recovering].

Вид доступу: тільки читання (RO).

Розмір: 1 сектор (512 байт).

Примітка: підтримується тільки 28-бітна адресація секторів (28-bit LBA).

Даний журнал містить список LBA-адрес секторів, на яких була зафіксована й зігнорована некоректуєма помилка ECC при виконанні операції READ CONTINUOUS (див. AV feature). При цьому, виконання процедури автоматичного перепризначення збійного сектора (ADR – Automatic Defects

Reassigment) накопичувачем заблоковано. Журнал може містити до 126 записів. Даний журнал доступний для читання тільки при дозволеній операції READ CONTINUOUS. У противному випадку накопичувач поверне код помилки ERR->ABRT, перерве виконання команди або поверне порожній журнал. Після успішного читання журналу, у самому накопичувачі він буде очищений.

Reassigned Sector Log

[under construction]

Drive Activity Log

[under construction]

Drive Time Buffer Log

[under construction]

Host Vendor Specific Log

Тип: Користувальницькі журнали.

Вид доступу: читання/запис (R/W).

Розмір: максимум 31 журнал по 16 секторів (253,952 байт).

Примітка: зміст і формат журналу – кожне, на розсуд користувача.

Цей вид журналу може бути використаний для зберігання довільних користувальницьких даних. Для запису цього журналу використовується команда WRITE SMART LOG. Якщо даний журнал жодного разу не був записаний, то при читанні накопичувач поверне порожній журнал, заповнений нулями.

Device Vendor Specific Log

Тип: Технічні журнали виробника.

Вид доступу: не визначений, на розсуд виробника (VS).

Розмір: максимум 31 журнал по 16 секторів (253,952 байт).

Примітка: зміст, формат і розміри журналу – на розсуд виробника.

Цей вид журналу призначений для внутрішнього використання фірмовими утилітами виробника, для зберігання результатів роботи убудованих підпрограм аналізу й діагностики стану накопичувача й т.п. Можливість читання/запису цього виду журналу встановлюється виробником і не документується. Нові

накопичувачі Seagate (моделі Ux і Barracuda ATA) підтримують і навіть реально використовують ще три види журналів SMART, однак їхнє призначення й опис поки не відомі.

Вбудовані функції самоконтролю (self-test)

Практично з моменту появи стандарту S.M.A.R.T. II, у більшості накопичувачів з'явилася нова функція – внутрішня діагностика й самоконтроль, для поглибленого контролю стану механіки накопичувача, поверхні дисків і т.п. Для запуску цієї функції, у набір команд S.M.A.R.T. була введена нова команда – SMART EXECUTE OFF-LINE IMMEDIATE. Результат роботи зберігається або в спеціалізованих атрибутах, або окремим параметром серед інших даних в атрибутах. Якщо накопичувач підтримує журнали S.M.A.R.T., то результат виконання тестів зберігається також у журналі Self-test Log. Після виконання тесту, накопичувач в обов'язковому порядку обновляє показання у всіх атрибутах і інших параметрах. Якщо під час виконання внутрішнього тесту накопичувач одержить по інтерфейсу нову команду, то виконання тесту переривається й накопичувач приступає до обробки команди, що надійшла.

Методи тестування

Існує два способи запуску тестів S.M.A.R.T.: автономний (off-line) або монопольний (captive). Результат тесту завжди зберігається накопичувачем у даних S.M.A.R.T. При автономному запуску накопичувач повідомляє про успішне завершення команди ДО її ФАКТИЧНОГО виконання й тільки після цього виконує тест. При цьому, по інтерфейсу прапор ЗАЙНЯТИЙ (BSY) не виставляється й накопичувач у будь-який момент готовий приступитися до виконання чергової інтерфейсної команди, припиняючи роботу тесту. Фактично, тест виконується у фоновому режимі. При запуску тесту в монопольному режимі, по інтерфейсу виставляється прапор ЗАЙНЯТИЙ (BSY) і накопичувач починає безпосереднє виконання тесту в режимі реального часу. Будь-яка інтерфейсна команда під час виконання цього тесту приведе до його переривання й зупинки, після чого накопичувач приступиться до обробки команди, що надійшла.

Різновиди тестів S.M.A.R.T.

Офіційно документовані три види внутрішніх тестів, однак ще існує набір так званих "активних" тестів, функціональні особливості яких різні в різних виробників і для широкої публіки не документовані.

Таблиця 3.2 – Різновиди тестів

№	Назва тесту	off-line	captive
1	Off-line collection	+	-
2	Short Self-test	+	+
3	Extended Self-test	+	+
4	Drive Activity test #1..#4	+	+

Час тестування може варіюватися від 1 секунди (Quantum) до 54 мінут (Fujitsu MPG3409AT). Підтримка першого тесту найбільш імовірна навіть у дуже старих накопичувачах 4-5 літні давнини. Другий і третій тести з'явилися відносно недавно, як данина впровадженим складним технологічним рішенням – для повного контролю стану накопичувача довелося реалізовувати більше глибокі й точні тести. Підтримка 4-х "активних" тестів (див. таблицю, п.4) офіційно не документована.

Реальний набір виконуваних тестами функцій можна розглянути на прикладі тестів, підтримуваних жорсткими дисками Hitachi.

Таблиця 3.3 – Реальний набір виконуваних тестами функцій

Функція тесту	Short Self test	Extended Self test	Off-line Collection
Raw Read Error Rate Test	YES	YES	YES
Write Test	YES	YES	NO
Servo Test	YES	YES	NO
Partial Read Scanning	YES	NO	NO
Full Read Scanning	NO	YES	YES

3.2 Розробка структурної схеми

На рисунку 3.1 зображена структурна схема розробленої системи тестування жорсткого диску з використанням технології SMART.

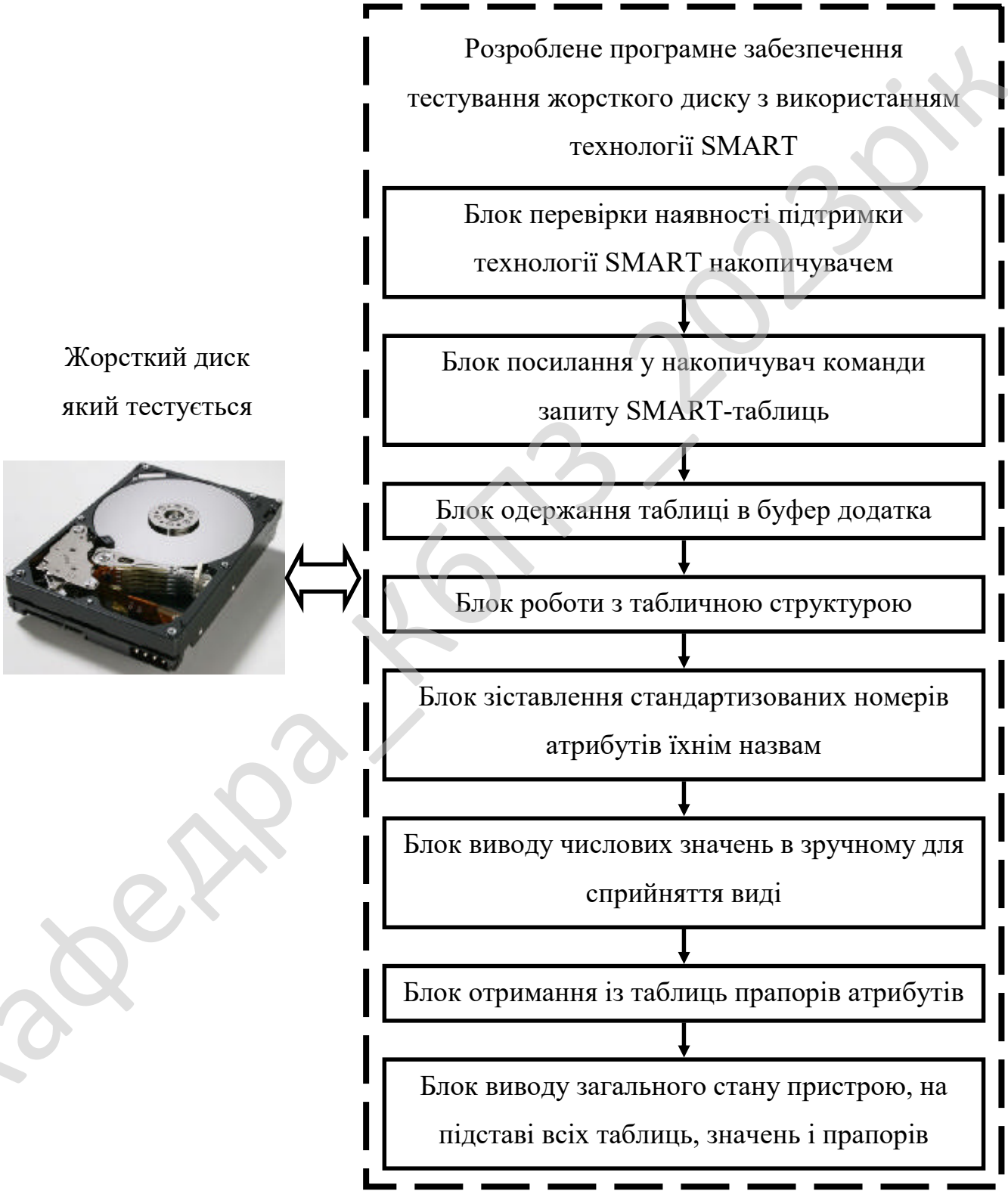

Рисунок 3.1 – Структурна схема системи

Структурна схема складається з наступних блоків:

– Блок перевірки наявності підтримки технології SMART накопичувачем.

– Блок посилання у накопичувач команди запиту SMART-таблиць.

– Блок одержання таблиці в буфер додатка.

– Блок роботи з табличною структурою, що витягає з них номера атрибутів і їхні числові значення.

– Блок зіставлення стандартизованих номерів атрибутів їхнім назвам (іноді – залежно від типу, моделі або фірми-виробника HDD).

– Блок виводу числових значень в зручному для сприйняття виді.

– Блок отримання із таблиць прапорів атрибутів (ознаки, що характеризують призначення атрибута в рамках конкретної firmware накопичувача, наприклад, «життєво важливий» або «лічильник»).

– Блок виводу загального стану пристрою, на підставі всіх таблиць, значень і прапорів.

3.3 Розробка функціональної схеми

На рисунку 3.2 зображена функціональна схема системи. Нижче розглянемо її більш докладно.

Функціональна схема складається з наступних блоків:

1. Блок читання атрибутів:

– Частота появи помилок при читанні даних з диска.

– Середня продуктивність (пропускна здатність) диска.

– Час розкручування шпинделя.

– Кількість циклів запуск/останов шпинделя.

– Кількість перепризначених секторів.

– Запас каналу читання.

– Частота появи помилок позиціонування БМГ.

– Середня продуктивність операцій позиціонування БМГ.

– Кількість відпрацьованих годин у включеному стані.

– Кількість повторів спроб старту шпинделя диска.

– Кількість повторів спроб рекалібровки накопичувача.

– Кількість повних циклів запуску/останова жорсткого диска.

– Частота появи "програмних" помилок при читанні даних з диска.

– Кількість циклів виводу БМГ у спеціальну парковочну зону/у робоче положення.

– Температура.

– Кількість операцій перепризначення (ремаппінгу).

– Поточна кількість нестабільних секторів.

– Кількість нескоректованих помилок.

– Загальна кількість помилок CRC у режимі UltraDMA.

– Частота появи помилок при записі даних.

– Зрушення пакета дисків щодо осі шпинделя.

– Частота появи помилок у результаті ударних навантажень.

– Навантаження на привод БМГ, викликана загальним наробітком годин накопичувачем.

– Навантаження на привод БМГ, викликана численними повтореннями операцій читання, запису, позиціонування головок і т.п.

– Навантаження на привод БМГ, викликане тертям у механічних частинах накопичувача.

– Загальна кількість циклів навантаження на привод БМГ.

– Загальний час навантаження на привод БМГ.

– Кількість зусиль обертаючого моменту привода.

– Кількість зафіксованих повторів включення/вимикання живлення накопичувача.

– Амплітуда тремтіння головок (GMR-head) у робочому стані.

Рисунок 3.2 – Функціональна схема системи

2. Блок читання типів атрибутів:

– Pre-failure (PF). Якщо атрибут має цей тип, то поле threshold атрибута містить мінімально припустиме значення атрибута, нижче якого не гарантується працездатність накопичувача й різко збільшується ймовірність його виходу з ладу.

– On-line collection (OC). Указує, що значення даного атрибута обновляється (обчислюється) під час виконання on-line тестів S.M.A.R.T. або ж під час обох видів тестів (on-line/off-line). У противному випадку, значення атрибута обновляється тільки при виконанні off-line тестів.

– Performance related (PR). Указує на те, що значення цього атрибута прямо залежить від продуктивності накопичувача за окремими показниками (seek/throughput/etc. performance). Звичайно обновляється після виконання self-test'ов SMART.

– Error rate (ER). Указує на те, що значення атрибута відбиває відносну частоту помилок по даному параметрі (raw read/write, seek, etc.).

– Events count (EC). Указує на те, що атрибут є лічильником подій.

– Self-preserve (SP). Указує на те, що значення атрибута обновляється й зберігається автоматично (звичайно при кожному старті накопичувача й при виконанні тестів SMART).

3. Блок автономного сканування поверхні.

4. Блок читання журналу помилок:

– Каталог журналів S.M.A.R.T.

– Сумарний журнал помилок.

– Комплексний журнал помилок.

– Розширений комплексний журнал помилок.

– Журнал результатів самоконтролю.

– Розширений журнал результатів самоконтролю.

– Журнал параметрів продуктивності потоків.

– Журнал помилок потокового запису.

– Журнал помилок потокового читання.

– Журнал непоправних помилок.

– Користувальницькі журнали.

– Технічні журнали виробника.

5. Блок вбудованих функцій самоконтролю.

6. Блок вибору методів тесту:

– автономний (off-line);

– монопольний (captive).

7. Блок набору «активних» тестів.

Розглянувши усі блоки функціональної схеми перейдемо до розгляду діаграми взаємодії процесів, які відбуваються у системі.

3.4 Розробка діаграми процесів

Діаграма взаємодії процесів системи, розробленої у результаті виконання бакалаврського проектування, наведена на рисунку 3.3.

Усі процеси, які є у системі взаємодіють наступним чином.

Спершу запускається процес початку/кінця програми.

Він взаємодіє з наступними процесами:

– Процесом перевірки наявності підтримки технології SMART.

– Процесом виведення результатів.

Перший процес, тобто процес перевірки наявності підтримки технології SMART, взаємодіє з процесом запиту SMART-таблиць.

Цей процес взаємодіє з процесом отримання із таблиць прапорів атрибутів.

Процес отримання із таблиць прапорів атрибутів, взаємодіє з процесом одержання таблиці у буфер додатка.

Як тільки таблиця одержується у буфер додатка, то запускаються наступні процеси:

– Процес зіставлення номерів атрибутів їхнім назвам.
– Процес пошуку несправностей.
– Процес аналізу загального стану пристрою.
– Процес виведення результатів.

Останній процес є завершуючим у системі.

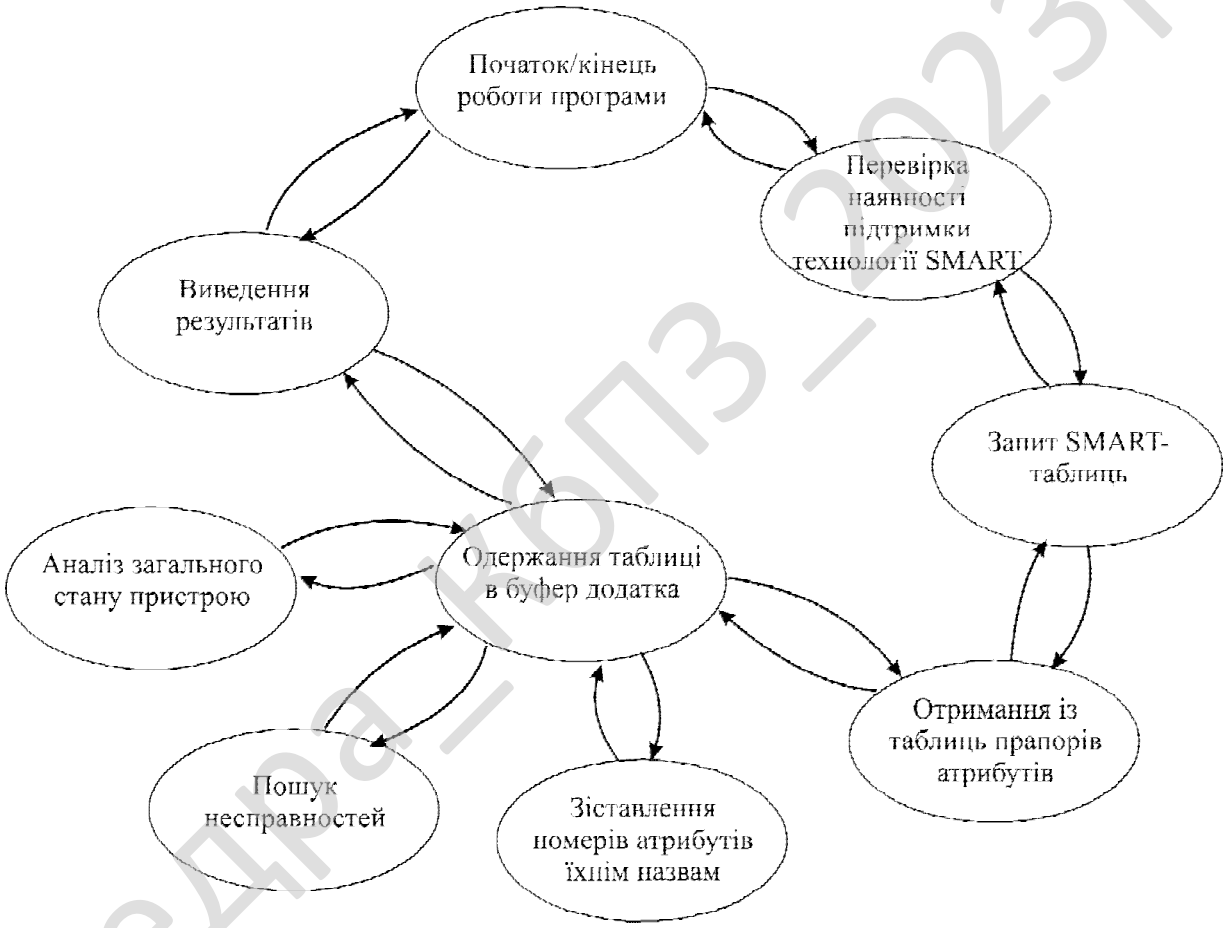

Рисунок 3.3 – Діаграма процесів системи

Таким чином, розглянувши опис системи, структурну, функціональну схеми системи, та діаграму взаємодії процесів перейдемо до опису блок-схем основної програми, та підпрограм, які використовуються, для реалізації системи.

4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ВІРНІСТЬ ПРОЕКТНИХ ТА ПРОГРАМНИХ РІШЕНЬ

4.1 Блок-схеми та опис алгоритмів функціонування системи

На рисунку 4.1 наведено блок-схему основної програми. Її робота складається з виконання наступних кроків.

Спершу відбувається виведення основного вікна програми.

Після цього ініціалізуються регістри контролерів жорстких дисків.

Наступним етапом, є зчитування значень регістрів.

Після зчитування значень регістрів відбувається їх обробка.

За цією дією слідує вибір першого диску зі списку.

Якщо виникає необхідність про обрання іншого диску зі списку, то користувач обирає інший диск.

По обраному диску виводиться наступна інформація:

– Інформація про вибраний пристрій.

– Інформація про поточний стан вибраного пристрою.

– Інформація про поточну температуру пристрою.

– Інформація про значення атрибутів SMART.

Після виводу вищеперерахованої інформації користувач обирає закінчувати йому роботу з програмний забезпеченням, або продовжувати її.

На рисунку 4.2 зображена блок-схема роботи підпрограми, яка реалізує технологію SMART.

Підпрограма починає роботу з перевірки наявності підтримки технології SMART накопичувачем. Якщо накопичувач не підтримує технологію SMART, то програма видає повідомлення про те, що пристрій не підтримує дану технологію, й на цьому завершується робота програми.

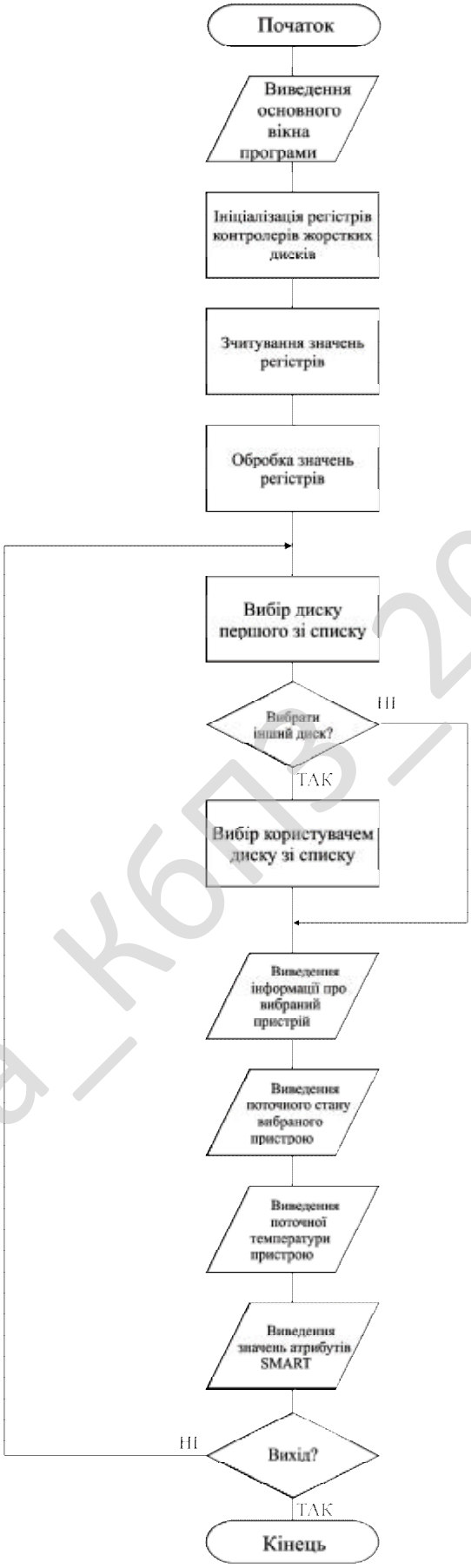

Рисунок 4.1 – Блок-схема роботи основної програми

У протилежному випадку, тобто, коли пристрій підтримує SMART, програмний продукт виконує наступні дії:

– У накопичувач посилається команда запиту SMART-таблиць.

– Таблиця одержується у буфер додатка.

– Відбувається робота з табличною структурою.

– Стандартизовані номери атрибутів зіставляються їхнім назвам.

– Виводяться числові значення в зручному для сприйняття виді.

– Отримуються прапори атрибутів із таблиць.

– Виводиться загальний стан пристрою на підставі всіх таблиць значень й прапорів.

На цьому підпрограма завершує свою роботу.

Наведемо короткий опис відомих атрибутів:

– * – (використовується в програмі HDD Speed) – даний покажчик показує, що відповідний атрибут S.M.A.R.T. є критичним для нормального функціонування накопичувача.

– Raw Read Error Rate – Частота появи помилок при читанні даних з диска.

– Throughput Performance – Середня продуктивність (пропускна здатність) диска.

– Spin Up Time – Час розкручування шпинделя.

– Start/Stop Count – Кількість циклів запуск/останов шпинделя.

– Reallocated Sectors Count – Кількість перепризначених секторів.

– Read Channel Margin – Запас каналу читання.

– Seek Error Rate – Частота появи помилок позиціонування блоку магнітних головок (БМГ).

– Seek Time Performance – Середня продуктивність операцій позиціонування БМГ.

– Power-On Hours – Кількість відпрацьованих годин у включеному стані.

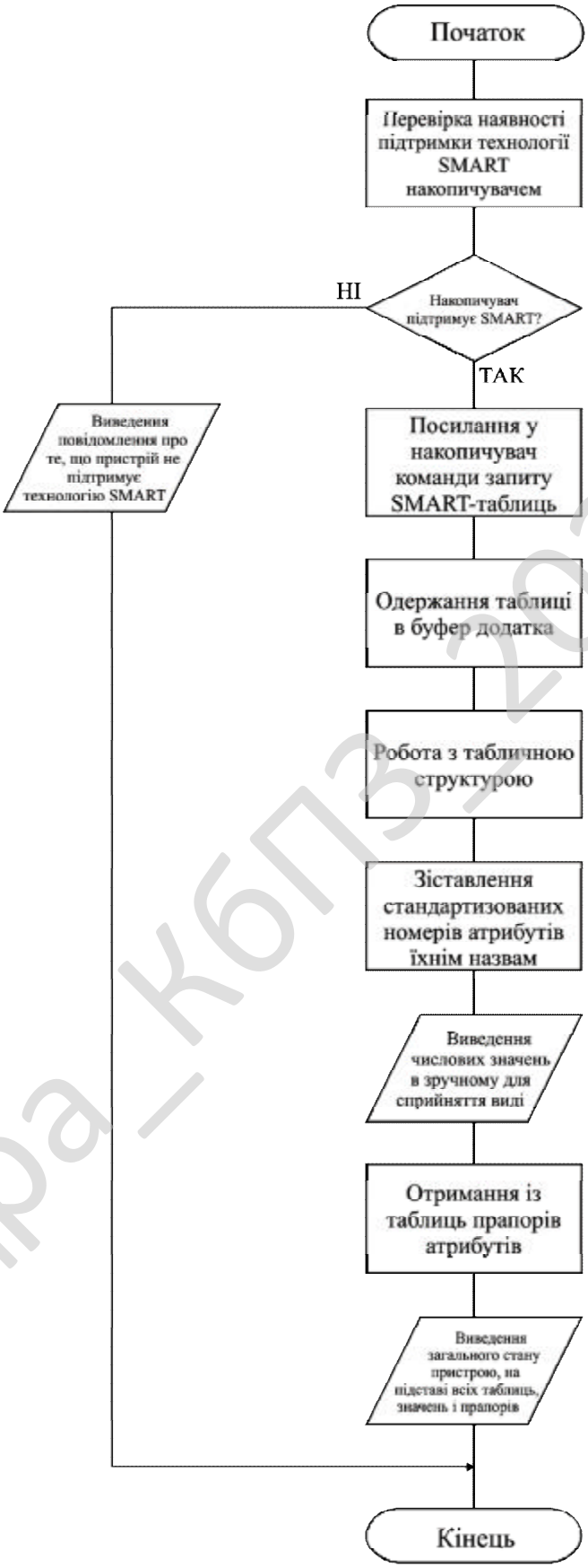

Рисунок 4.2 – Блок-схема роботи підпрограми, яка реалізує технологію SMART

– Spin Retry Count – Кількість повторів спроб старту шпинделя диска.

– Recalibration Retries – Кількість повторів спроб рекалібровки накопичувача.

– Device Power Cycle Count – Кількість повних циклів запуску/останова жорсткого диска.

– Soft Read Error Rate – Частота появи "програмних" помилок при читанні даних з диска.

– Load/Unload Cycle Count – Кількість циклів виводу БМГ у спеціальну парковочну зону/у робоче положення.

– Temperature – Температура.

– Reallocation Event Count – Кількість операцій перепризначення (ремаппінгу).

– Current Pending Sector Count – Поточна кількість нестабільних секторів.

– Uncorrectable Sector Count – Кількість нескоректованих помилок.

– UltraDMA CRC Error Count – Загальна кількість помилок CRC у режимі UltraDMA.

– Write Error Rate (Multi Zone Error Rate) – Частота появи помилок при записі даних.

– Disk Shift – Зрушення пакета дисків щодо осі шпинделя.

– G-Sense Error Rate – Частота появи помилок у результаті ударних навантажень.

– Loaded Hours – Навантаження на привод БМГ, викликана загальним наробітком годин накопичувачем.

– Load/Unload Retry Count – Навантаження на привод БМГ, викликана численними повтореннями операцій читання, запису, позиціонування головок і т.п.

– Load Friction – Навантаження на привод БМГ, викликане тертям у механічних частинах накопичувача.

– Load/Unload Cycle Count – Загальна кількість циклів навантаження на привод БМГ.

– Load-in Time – Загальний час навантаження на привод БМГ.

– Torque Amplification Count – Кількість зусиль обертаючого моменту привода.

– Power-Off Retract Count – Кількість зафіксованих повторів включення/вимикання живлення накопичувача.

– GMR Head Amplitude – Амплітуда тремтіння головок (GMR-head) у робочому стані.

Типи атрибутів

Кожний атрибут може мати деякий набір прапорів, що визначають його функціональні особливості. Нижче приводяться всі шість основних типів і їхні короткі описи:

– Pre-failure (PF). Якщо атрибут має цей тип, то поле threshold атрибута містить мінімально припустиме значення атрибута, нижче якого не гарантується працездатність накопичувача й різко збільшується ймовірність його виходу з ладу.

– On-line collection (OC). Указує, що значення даного атрибута обновляється (обчислюється) під час виконання on-line тестів S.M.A.R.T. або ж під час обох видів тестів (on-line/off-line). У противному випадку, значення атрибута обновляється тільки при виконанні off-line тестів.

– Performance related (PR). Указує на те, що значення цього атрибута прямо залежить від продуктивності накопичувача за окремими показниками (seek/throughput/etc. performance). Звичайно обновляється після виконання self-test'ов SMART.

– Error rate (ER). Указує на те, що значення атрибута відбиває відносну частоту помилок по даному параметрі (raw read/write, seek, etc.).

– Events count (EC). Указує на те, що атрибут є лічильником подій.

– Self-preserve (SP). Указує на те, що значення атрибута обновляється й зберігається автоматично (звичайно при кожному старті накопичувача й при виконанні тестів SMART).

Наведемо частини програмного кода, які реалізують деякі з вищеперерахованих функцій.

Функція читання та розшифровки таблиці атрибутів.

```
DWORD CAtaSmart::CheckSmartAttributeUpdate(DWORD index, SMART_ATTRIBUTE* pre,
SMART_ATTRIBUTE* cur)
{
     if(memcmp(pre, cur, sizeof(SMART_ATTRIBUTE) * MAX_ATTRIBUTE) == 0)
     {
          return SMART_STATUS_NO_CHANGE;
     }
     else
     {
          for(int i = 0; i < MAX_ATTRIBUTE; i++)
          {
               switch(cur[i].Id)
               {
               case 0x09: // Кількість відпрацьованих годин у включеному стані
                    {
                         DWORD preRawValue = MAKELONG(
                    MAKEWORD(pre[i].RawValue[0], pre[i].RawValue[1]),
                    MAKEWORD(pre[i].RawValue[2], pre[i].RawValue[3])
                              );
                         DWORD curRawValue = MAKELONG(
                    MAKEWORD(cur[i].RawValue[0], cur[i].RawValue[1]),
                    MAKEWORD(cur[i].RawValue[2], cur[i].RawValue[3])
                              );
     if(GetPowerOnHours(preRawValue, vars[index].DetectedTimeUnitType)
                         != GetPowerOnHours(curRawValue,
vars[index].DetectedTimeUnitType))
                         {
                              memcpy(pre, cur, sizeof(SMART_ATTRIBUTE) *
MAX_ATTRIBUTE);
                              return SMART_STATUS_MAJOR_CHANGE;
                         }
     if(GetPowerOnHours(preRawValue, vars[index].MeasuredTimeUnitType)
```

```
                        != GetPowerOnHours(curRawValue,
vars[index].MeasuredTimeUnitType))
                        {
                             memcpy(pre, cur, sizeof(SMART_ATTRIBUTE) *
MAX_ATTRIBUTE);
                             return SMART_STATUS_MAJOR_CHANGE;
                        }
                   }
                   break;
              case 0x0C: // Кількість зафіксованих повторів включення/вимикання
живлення накопичувача
                   {
                        DWORD preRawValue = MAKELONG(
                             MAKEWORD(pre[i].RawValue[0], pre[i].RawValue[1]),
                             MAKEWORD(pre[i].RawValue[2], pre[i].RawValue[3])
                             );
                        DWORD curRawValue = MAKELONG(
     MAKEWORD(cur[i].RawValue[0], cur[i].RawValue[1]),
                             MAKEWORD(cur[i].RawValue[2], cur[i].RawValue[3])
                             );
                        if(preRawValue != curRawValue)
                        {
                             memcpy(pre, cur, sizeof(SMART_ATTRIBUTE) *
MAX_ATTRIBUTE);
                             return SMART_STATUS_MAJOR_CHANGE;
                        }
                   }
                   break;
              case 0xC2: // Температура
                   if(pre[i].RawValue[0] != cur[i].RawValue[0]
                   || pre[i].CurrentValue != cur[i].CurrentValue)
                   {
                        memcpy(pre, cur, sizeof(SMART_ATTRIBUTE) * MAX_ATTRIBUTE);
                        return SMART_STATUS_MAJOR_CHANGE;
                   }
                   break;
              default:
                   break;
              }
         }
         return SMART_STATUS_MINOR_CHANGE;
     }
```

```
}
// Визначення виробника
          if(GetDiskInfo(i, -1, -1, INTERFACE_TYPE_UNKNOWN, VENDOR_UNKNOWN))
          {
               int index = (int)vars.GetCount() - 1;
               CString cmp;
               cmp = vars[index].Model;
               if(cmp.Find(_T("DW C")) == 0 // WDC
               || cmp.Find(_T("iHat")) == 0 // Hitachi
               || cmp.Find(_T("ASSM")) == 0 // SAMSUNG
               || cmp.Find(_T("aMtx")) == 0 // Maxtor
               || cmp.Find(_T("OTHS")) == 0 // TOSHIBA
               || cmp.Find(_T("UFIJ")) == 0 // FUJITSU
               )
               {
                    vars[index].SerialNumber = vars[index].SerialNumberReverse;
                    vars[index].FirmwareRev = vars[index].FirmwareRevReverse;
                    vars[index].Model = vars[index].ModelReverse;
                    vars[index].ModelSerial = vars[index].Model +
vars[index].SerialNumber;
                    vars[index].ModelSerial.Replace(_T("/"), _T(""));
               }
          }
// Розмір диску та розмір буферу
     asi.Cylinder = identify->LogicalCylinders;
     asi.Head = identify->LlogicalHeads;
     asi.Sector = identify->LogicalSectors;
     asi.Sector28 = identify->TotalAddressableSectors;
     asi.Sector48 = identify->MaxUserLba;
     asi.DiskSizeChs   = (DWORD)(((ULONGLONG)identify->LogicalCylinders *
identify->LlogicalHeads * identify->LogicalSectors * 512) / 1000 / 1000  - 50);
     asi.DiskSizeLba28 = (DWORD)(((ULONGLONG)identify->TotalAddressableSectors *
512) / 1000 / 1000 - 50);
     if(asi.IsLba48Supported)
     {
          asi.DiskSizeLba48 = (DWORD)(((ULONGLONG)identify->MaxUserLba * 512) /
1000 / 1000 - 50);
     }
     asi.BufferSize = identify->BufferSize * 512;
     if(asi.IsNvCacheSupported)
     {
          asi.NvCacheSize = identify->NvCacheSizeLogicalBlocks * 512;
```

```
    }
    if(asi.DiskSizeChs == 0)
    {
        asi.TotalDiskSize = 0;
    }
    else if(asi.DiskSizeLba48 > asi.DiskSizeLba28)
    {
        asi.TotalDiskSize = asi.DiskSizeLba48;
    }
    else if(asi.DiskSizeLba28 > asi.DiskSizeChs)
    {
        asi.TotalDiskSize = asi.DiskSizeLba28;
    }
    else
    {
        asi.TotalDiskSize = asi.DiskSizeChs;
    }
    // Перевірка помилок для контролеру External ATA
    if(asi.IsLba48Supported && (identify->TotalAddressableSectors < 268435455 &&
asi.DiskSizeLba28 != asi.DiskSizeLba48))
    {
        asi.DiskSizeLba48 = 0;
    }
// перевірка підтримки S.M.A.R.T.
    switch(asi.CommandType)
    {
    case CMD_TYPE_PHYSICAL_DRIVE:
// РОЗШИФРУВАННЯ ЗНАЧЕНЬ
//        SendAtaCommandPd(physicalDriveId, 0xEF, 0x42, 0xFE);
//        SendAtaCommandPd(physicalDriveId, 0xEF, 0x05, 0xFE);
        if(GetSmartAttributePd(physicalDriveId, &asi))
        {
            GetSmartThresholdPd(physicalDriveId, &asi);
            asi.DiskStatus = CheckDiskStatus(asi.Attribute, asi.Threshold,
asi.AttributeCount, asi.VendorId);
            asi.IsSmartEnabled = TRUE;
        }
        else if(ControlSmartStatusPd(physicalDriveId, ENABLE_SMART))
        {
    if(GetSmartAttributePd(physicalDriveId, &asi))
            {
    GetSmartThresholdPd(physicalDriveId, &asi);
```

```
                    asi.DiskStatus = CheckDiskStatus(asi.Attribute, asi.Threshold,
asi.AttributeCount, asi.VendorId);
                    asi.IsSmartEnabled = TRUE;
               }
          }
          break;
     case CMD_TYPE_SCSI_MINIPORT:
// РОЗШИФРУВАННЯ ЗНАЧЕНЬ
//        SendAtaCommandScsi(scsiPort, scsiTargetId, 0xEC, 0x00, 0x00); // ID
//        SendAtaCommandScsi(scsiPort, scsiTargetId, 0xEF, 0x05, 0x80); // APM
//        SendAtaCommandScsi(scsiPort, scsiTargetId, 0xEF, 0x42, 0x80); // AAM
// Підтримка функцій
/*-----------------------------------------------*/
DWORD CAtaSmart::CheckDiskStatus(SMART_ATTRIBUTE* attribute, SMART_THRESHOLD*
threshold, DWORD attributeCount, DWORD vendorId)
{
     int error = 0;
     int caution = 0;
     BOOL flagUnknown = TRUE;
     for(DWORD j = 0; j < attributeCount; j++)
     {
          if( attribute[j].Id != 0xBE // Температура воздуха
          &&   threshold[j].ThresholdValue != 0
          &&   attribute[j].CurrentValue <= threshold[j].ThresholdValue)
          {
               error++;
          }
          switch(attribute[j].Id)
          {
          case 0x05: // Кількість перепризначених секторів
//        case 0xC4: // Кількість операцій перепризначення (ремаппінгу).
          case 0xC5: // Поточна кількість нестабільних секторів
          case 0xC6: // Кількість нескоректованих помилок
               if(attribute[j].RawValue[0] == 0xFF
               && attribute[j].RawValue[1] == 0xFF
               && attribute[j].RawValue[2] == 0xFF
               && attribute[j].RawValue[3] == 0xFF)
               {
                    flagUnknown = FALSE;
               }
               else
               {
```

```
                    caution += attribute[j].RawValue[0]
                              + attribute[j].RawValue[1];
                    flagUnknown = FALSE;
               }
               break;
          case 0xBB: // Визначення фірми-розробника
               if(vendorId == VENDOR_MTRON)
               {
                    if(attribute[j].CurrentValue == 0)
                    {
                         error = 1;
                    }
                    else if(attribute[j].CurrentValue < 10)
                    {
                         caution = 1;
                    }
                    else
                    {
                         flagUnknown = FALSE;
                    }
               }
               break;
          default:
               break;
          }
     }
     if(error > 0)
     {
          return DISK_STATUS_BAD;
     }
     else if(flagUnknown)
     {
          return DISK_STATUS_UNKNOWN;
     }
     else if(caution > 0)
     {
          return DISK_STATUS_CAUTION;
     }
     else
     {
          return DISK_STATUS_GOOD;
     }
}
```

4.2 Захист розробленого програмного забезпечення

Tiny Encryption Algorithm (TEA) [1] – блочний алгоритм шифрування типу «Мережі Фейстеля». Алгоритм був розроблений на факультеті комп'ютерних наук Кембриджського університету Девідом Вілером (David Wheeler) і Роджером Нідгемом (Roger Needham) та вперше представлений в 1994 році [2] на симпозіумі зі швидкими алгоритмами шифрування в Льовені (Бельгія).

Шифр не патентований, широко використовується в ряді криптографічних додатків і широкому спектрі апаратного забезпечення, завдяки вкрай низькими вимогами до пам'яті й простоті реалізації. Алгоритм має як програмну реалізацію на різних мовах програмування, так і апаратну реалізацію на інтегральних схемах типу FPGA.

Захист розробленого програмного забезпечення буде відбуватися за допомогою алгоритму TEA, який заснований на бітових операцій з 64-бітним блоком, має 128-бітний ключ шифрування. Стандартна кількість раундів мережі Фейстеля біля 64 (32 циклу), однак, для досягнення найкращої продуктивності або шифрування, число циклів можна варіювати від 8 (16 раундів) до 64 (128 раундів). Мережа Фейстеля несиметрична через використання в якості операції накладення додавання по модулю 232.

Перевагами шифру є його простота в реалізації, невеликий розмір коду й досить висока швидкість виконання, а також можливість оптимізації виконання на стандартних 32-бітних процесорах, так як в якості основних операцій використовуються операції виключна «АБО» (XOR), побітового зсуву й додавання по модулю 232. Оскільки алгоритм не використовує таблиць підстановки і раундова функція досить проста, алгоритму потрібно не менше 16 циклів (32 раундів) для досягнення ефективної дифузії, хоча повна дифузія досягається вже через 6 циклів (12 раундів).

Алгоритм має відмінну стійкість до лінійного криптоаналізу і досить гарну до диференціального криптоаналізу. Головним недоліком цього алгоритму

шифрування є його вразливість до атак «на пов'язаних ключах» (англ. Related-key attack). Через простий розклад ключів кожен ключ має 3 еквівалентних ключа. Це означає, що ефективна довжина ключа складає всього 126 біт [3] [4], тому даний алгоритм не слід використовувати в якості геш-функції.

Опис алгоритму

Вихідний текст розбивається на блоки по 64 біта кожен. 128-бітний ключ K ділиться на чотири 32-бітних підключа K [0], K [1], K [2] і K [3]. На цьому підготовчий процес закінчується, після чого кожен 64-бітний блок шифрується протягом 32 циклів (64 раундів) за нижченаведеним алгоритмом. [5]

Припустимо, що на вхід n-го раунду ($1 \leq n \leq 64$) надходять права й ліва частини (L_n, R_n), тоді на виході n-го раунду будуть ліва й права частини (L_{n+1}, R_{n+1}), які обчислюються за такими правилами:

$$L_{n+1} = R_n.$$

Якщо $n = 2 * i - 1$ для $1 \leq i \leq 32$ (непарні раунди), то:

$$R_{n+1} = L_n (\{ [R_n\ 4] K [0] \} \{ R_n\ i * \delta \} \{ [R_n\ 5] K [1] \}).$$

Якщо $n = 2 * i$ для $1 \leq i \leq 32$ (парні раунди), то:

$$R_{n+1} = L_n (\{ [R_n\ 4] K [2] \} \{ R_n\ i * \delta \} \{ [R_n\ 5] K [3] \}).$$

де

X Y – операція додавання чисел X і Y по модулю 232.

X Y – побітове виключне АБО» (XOR) чисел X і Y, яке в мові програмування Сі позначається як X ^ Y

X Y і X Y – операції побітового зсуву числа X на Y біт вліво й вправо відповідно.

Константа δ була виведена з Золотого перерізу:

$$\delta = (-1) * 2^{31} = 2654435769 = 9E3779B9_h.$$

У кожному раунді константа множиться на номер циклу i. Це було зроблено для запобігання простих атак, заснованих на симетрії раундів.

Також очевидно, що в алгоритмі шифрування TEA немає як такого алгоритму розкладу ключів. Замість цього в непарних раундах використовуються підключі K [0] та [1], у парних – K [2] і [3].

Так як це блочний шифроалгоритм, де довжина блоку 64-біт, а довжина даних може бути не кратна 64-біт, значення всіх байтів, які доповнюють блок до кратності в 64-біт, встановлюється в 0x01.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

На рисунку 5.1 зображено інтерфейс основного вікна програми.

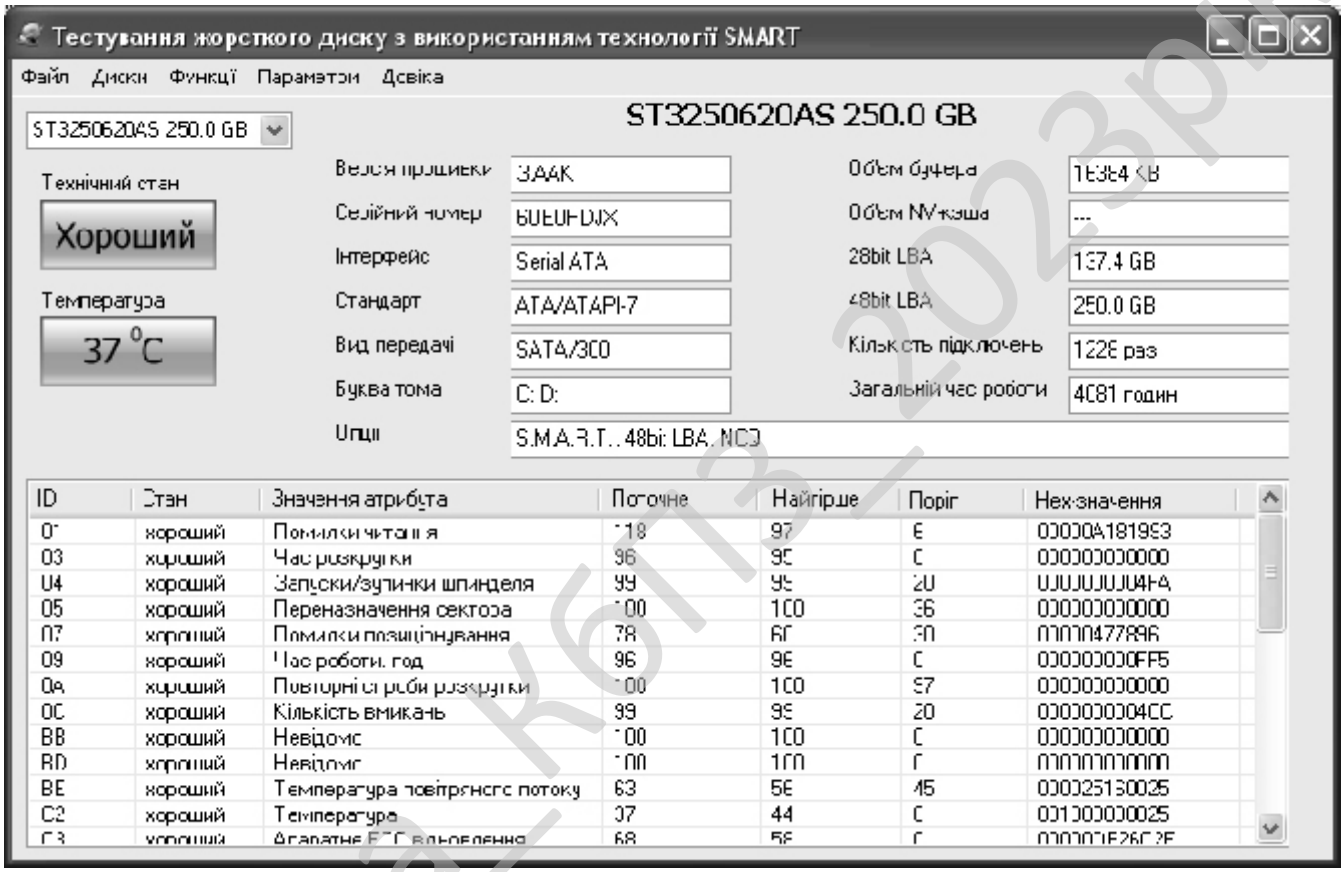

Рисунок 5.1 – Основне вікно програми

Як видно з рисунку, у програмі існують такі меню як:

– Файл – де відбуваються типові операції.

– Диски – де обираються диски з якими відбувається робота.

– Функції – де визначені усі функції, які може реалізувати програма.

– Параметри – де визначені усі параметри, які може задати користувач.

– Довідка – де знаходяться дані про роботу програми та розробника програми.

На головній формі реалізовані наступні поля виводу даних:

1. Поле вибору диску.
2. Поле виводу технічного стану.
3. Поле виводу температури.
4. Поле виводу версії прошивки.
5. Поле виводу серійного номеру.
6. Поле виводу типу інтерфейсу.
7. Поле виводу стандарту.
8. Поле виводу виду передачі.
9. Поле виводу опції.
10. Поле виводу об’єму буфера.
11. Поле виводу об’єму NV-кешу.
12. Поле виводу 28bit LBA.
13. Поле виводу 48bit LBA.
14. Поле виводу кількості підключень.
15. Поле виводу загального часу роботи.
16. Поле виводу значень атрибутів, яке містить у собі:

– ID атрибута.

– Стан атрибута.

– Значення атрибута.

– Поточне значення атрибута.

– Найгірше значення атрибута.

– Порогове значення атрибута.

– HEX-значення атрибута.

На рисунку 5.2 зображене вікно довідки, на якому наведено наступні дані:

– Тема програми.

– Розробник програми.

– Керівник роботи.

– Місце виконання програми.

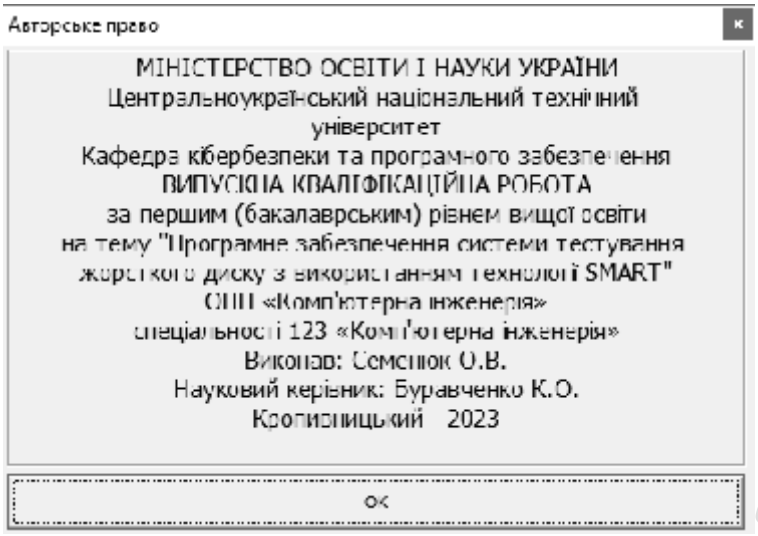

Рисунок 5.2 – Довідка

6 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти, призначено для системи тестування жорсткого диску з використанням технології SMART.

В межах України в недостатній мірі представлені вітчизняні розробки в цій області.

Рішення завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем тестування жорсткого диску з використанням технології SMART.

– Досліджена система тестування жорсткого диску з використанням технології SMART.

– На основі отриманих результатів досліджень створена програмна реалізація системи тестування жорсткого диску з використанням технології SMART.

Розроблені під час виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання тестування жорсткого диску з використанням технології SMART.

Розроблене програмне забезпечення має простий, дружній та зручний інтерфейс користувача, що забезпечує легкість у освоєнні роботи програмного продукту, зручність у використанні, і не потребує особливих спеціальних знань.

При створені програмного забезпечення було використано об’єктно-орієнтований підхід, що відповідає сучасним тенденціям у галузі розробки комерційних програмних систем.

Програма реалізована на мові високого рівня Visual C++. Дана мова програмування дозволяє найбільш ефективно обробляти дані призначені для системи тестування жорсткого диску з використанням технології SMART. Це

дозволило мінімізувати строк розробки програмного забезпечення, і, як слід, зменшити витрати на його розробку. Запропоноване програмне забезпечення ділиться на загальне програмне забезпечення, що поставляється із засобами обчислювальної техніки й спеціальне програмне забезпечення, що спеціально розроблене для даної конкретної системи й включає програми, що реалізують її функції.

Програма призначена для виконання під управлінням багатозадачної операційної системи Windows 10/11.

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм TEA.

В цілому створене програмне забезпечення підтверджує правильність використаних проектних рішень та повністю відповідає вимогам технічного завдання. Створене програмне забезпечення має потенційну можливість для подальшого вдосконалення і застосування у різних галузях.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Коваленко А.С. Разработка структуры базы данных интегрированной информационной системы / А.С. Коваленко, А.В. Коваленко // Информационные технологии и защита информации в информационно-коммуникационных системах: монографія / Под редакцией профессора В.С. Пономаренко. – Х.: Вид-во ТОВ «Щедра садиба плюс», 2015. – С. 54-64.

2. Кожанова А.С. Обґрунтування необхідності створення систем технічної діагностики інтегрованих інформаційних систем / О.А. Смірнов, А.С. Кожанова, О.В. Коваленко // Системи обробки інформації. – Х.: ХУПС, 2013. – Вип. 6(113). – С. 255-257.

3. Коваленко А.С. Задачи распознавания ситуаций в ERP системах / А.В. Коваленко ., А.А. Смірнов, А.С. Коваленко // Системи обробки інформації. – Х.: ХУПС, 2014. – Вип. 4(120). – С. 161-164.

4. Коваленко А.С. Підсистема технічної діагностики для автоматизації процесів керування в інтегрованих інформаційних системах / А.С. Коваленко , О.А.Смірнов, О.В. Коваленко // Системи озброєння і військова техніка.– Х.: ХУПС, 2014. – № 1(37). – С. 126-129.

5. Коваленко А.С. Анализ эффективности использования экспертной системы технической диагностики с традиционной структурой / А.С. Коваленко, А.А. Смирнов, А.В. Коваленко // Системи озброєння і військова техніка. – Х.: ХУПС, 2014. – № 2(38). – С. 106-108.

6. Коваленко А.С. Разработка структуры экспертной системы технической диагностики интегрированной информационной системы / А.С. Коваленко, А.А. Смирнов, А.В. Коваленко // Наука і техніка Повітряних Сил Збройних Сил України. – Харків: ХУПС, 2014. – № 2(15). – С.154-157.

7. Коваленко А.С. Разработка структуры экспертной системы технической диагностики интегрированной информационной системы / А.С. Коваленко,

А.А. Смирнов, А.В. Коваленко // Наука і техніка Повітряних Сил Збройних Сил України. – Харків: ХУПС, 2014. – № 2(15). – С.154-157.

8. Коваленко А.С. Структура системи технічної діагностики інтегрованих інформаційних систем / А.С. Коваленко, О.А. Смірнов, О.В. Коваленко // Збірник наукових праць Кіровоградського національного технічного університету / техніка в сільськогосподарському виробництві, галузеве машинобудування, автоматизація. – Кіровоград: Вид-во КНТУ, 2014. – Вип. 27. – С. 245-251.

9. Коваленко А.С. Дослідження будови інтегрованої інформаційної системи та її елементів / А.С. Коваленко, О.А. Смірнов, О.В. Коваленко // Системи озброєння і військова техніка. – Х.: ХУПС, 2014. – № 4(40). – С. 85-88.

10. Коваленко А.С. Розробка структури бази даних для обліку технічного стану елементів інтегрованої інформаційної системи з урахуванням вимог споживачів інформації / А.С. Коваленко, О.А. Смірнов, О.В. Коваленко // Системи обробки інформації. – Х.: ХУПС, 2015. – Вип. 1(126). – С. 75-79.

11. Коваленко А.С. Обґрунтування набору даних для оцінки технічного стану інтегрованої інформаційної системи / А.С. Коваленко, О.А. Смірнов, О.В. Коваленко // Збірник наукових праць Харківського університету Повітряних Сил. – Харків: ХУПС, 2015. – Вип. 1(42). – С.39-41.

12. Коваленко А.С. Експертна система технічного діагностування інтегрованої інформаційної системи / А.С. Коваленко, О.А. Смірнов, О.В. Коваленко // Системи озброєння і військова техніка. – Х.: ХУПС, 2015. – № 1(41). – С. 106-111.

13. Коваленко А.С. Удосконалення методу технічного обслуговування об’єктів інтегрованої інформаційної системи / А.С. Коваленко, О.А. Смірнов, О.В. Коваленко, О.П. Доренський // Системи озброєння і військова техніка. – Х.: ХУПС, 2016. – № 2(46). – С. 109-114.

14. Коваленко А.С. Метод визначення оптимального комплексу робіт з відновлення працездатності інтегрованої системи технічної діагностики в умовах

ресурсних обмежень / А.С. Коваленко // Системи обробки інформації. – Х.: ХУПС, 2016. – Вип. 3(140). – С. 69-72.

15. Kovalenko A.S. Information model and its element for displaying information on technical condition of objects of integrated information system / A.S. Kovalenko, A.A. Smirnov, A.V. Kovalenko, A.P. Dorensky // International Journal of Computational Engineering Research (IJCER). – India: Delhi, 2016. – Volume 6, Issue 1. – P. 21-27.

16. Кожанова А.С. Система технічної діагностики інтегрованих інформаційних систем – обґрунтування необхідності створення, визначення понятійного апарату та напрямів досліджень / А.С. Кожанова, О.А. Смірнов, М.П. Савченко, Д.М. Ізосімов, В.В. Мороз // Створення та модернізація озброєння і військової техніки в сучасних умовах: Тринадцята наук.-техн. конф., 5-6 вер. 2013 р., м. Феодосія: тези доп. – Феодосія: ДНВЦ, 2013. – С. 187-188.

17. Кожанова А.С. Визначення основних напрямків досліджень щодо створення системи технічної діагностики інтегрованих інформаційних систем / А.С. Кожанова, О.А. Смірнов, А.В. Челпанов // Проблемні питання розвитку озброєння та військової техніки Збройних Сил України: IV наук.-техн. конф., 16-20 груд. 2013 р., м. Київ: зб. тез. – Київ: ЦНДІ ОВТ ЗСУ, 2013. – С. 293.

18. Коваленко А.С. Обґрунтування необхідності створення систем технічної діагностики інтегрованих інформаційних систем / А.С. Коваленко, О.А. Смірнов, О.В. Коваленко // Інформатика та системні науки : V Всеукр. наук.-практ. конф., 13–15 бер. 2014 р., м. Полтава : зб. тез. – Полтава: ПУЕТ, 2014. – С. 292-294.

19. Коваленко А.С. Задачи распознавания ситуаций в системах организационной стратегии интеграции производства и операций / А.С. Коваленко, А.В. Коваленко // Комбінаторні конфігурації та їх застосування: XVI міжнар. наук.-практ. сем., 11-12 квіт. 2014 р., м. Кіровоград: зб. тез. – Кіровоград: КНТУ, 2014. – С. 53-55.

20. Коваленко А.С. Створення систем технічної діагностики для автоматизації процесів керування в інтегрованих інформаційних системах / А.С. Коваленко, О.А. Смірнов, О.В. Коваленко // Проблеми і перспективи розвитку ІТ-індустрії: VI між нар. наук.-практ. конф., 17-18 квіт. 2014 р., м. Харків: зб. тез. – Харків: ХНЕУ, 2014. – С. 241.

21. Коваленко А.С. Визначення понятійного апарату та напрямів досліджень для синтезу систем технічної діагностики інтегрованих інформаційних систем / А.С. Коваленко, О.А. Смірнов, О.В. Коваленко // Комп'ютерне моделювання у наукоємних технологіях (КМНТ-2014): наук.-техн. конф. з міжнар. участю, 28-31 трав. 2014 р., м. Харків: зб. наук. праць. – Харків: ХНУ, 2014. – С. 190-193.

22. Коваленко А.С. Основні складові та функції системи технічної діагностики інтегрованих інформаційних систем / Коваленко А.С. // Інформаційні технології та комп'ютерна інженерія: наук.-практ. конф., 4 груд. 2014 р., м. Кіровоград: зб. тез доп. – Кіровоград: КНТУ, 2014. – С. 236.

23. Коваленко А.С. Розробка структури бази даних інтегрованої інформаційної системи / А.С. Коваленко, О.А. Смірнов, О.В. Коваленко // Проблеми і перспективи розвитку ІТ-індустрії: VII міжнар. наук.-практ. конф., 17-18 квіт. 2015 р., м. Харків: зб. тез. – Харків: ХНЕУ, 2015. – С. 15.

24. Коваленко А.С. Дослідження елементів інтегрованої інформаційної системи / А.С. Коваленко, О.А. Смірнов, О.В. Коваленко // Комбінаторні конфігурації та їх застосування: XVII між нар. наук.-практ. сем., 17-18 квіт. 2015 р., м. Кіровоград: зб. тез – Кіровоград: КНТУ, 2015. – С. 5.

25. Коваленко А.С. Метод автоматизованої перевірки результатів вимірювання параметрів об'єкті в інтегрованої інформаційної системи / А.С. Коваленко, О.А. Смірнов, О.В. Коваленко // Стратегія якості у промисловості і освіті: XI міжнар. конф., 1 – 5 черв. 2015 р., м. Варна, Болгарія.: зб. матер. – Варна: ТУВ, 2015. – С. 423-426.

26. Коваленко А.С. Обґрунтування необхідності створення розподіленої бази даних для забезпечення захисту рухомих повітряних об`єктів / А.С. Коваленко, О.А. Смірнов, О.В. Коваленко // Перспективні напрями захисту інформації: І всеукр. наук.-практ. конф., 07 вер. 2015 р., м. Одеса: зб. тез доп. – Одеса: ОНАЗ, 2015. – С. 35-39.

27. Коваленко А.С. Розробка інформаційної моделі автоматизованої оцінки технічного стану інтегральної інформаційної системи / А.С. Коваленко, О.А. Смірнов, О.В. Коваленко // Інформаційні технології та взаємодії (ІТ & І): ІІ між нар. наук.-практ. конф., 3-5 лист. 2015 р., м. Київ: тези доп. – Київ: КНУ ім. Т. Шевченка, 2015. – С. 41-42.

28. Коваленко А.С. Разработка метода усовершенствования технического обслуживания интегрированной информационной системы / А.С. Коваленко, О.А. Смірнов, О.В. Коваленко // Информационные и телекоммуникационные технологии: образование, наука, практика: ІІ междунар. научн.-практ. конф., 3-4 дек. 2015 г., г. Алматы, Казахстан: сб. труд. – Алматы: КазНИТУ им. К.И. Сатпаева, 2015. – Т.2. – С. 423-427.

29. Королюк Н.А. Оценка временных интервалов работы лица, принимающего решение, на автоматизированном командном пункте / Н.А. Королюк, А.И. Тимочко // Системи обробки інформації. – Х.: ХУПС, 2005. – Вип. 8 (48). – С. 51-54.

30. Костерев В.В. Надёжность технических систем и управление риском: учебн. пособ. / В.В. Костерев. – М.: МИФИ, 2008. – 280 с.

31. Костюков А.В. Підвищення операційної ефективності підприємств на основі моніторингу в реальному часі. / А.В. Костюков, В.М. Костюков. – М.: Машинобудування, 2009. – 192 с.

32. Лазарев А.А. Выбор показателя затрат для анализа сравнительной экономической эффективности техники конечного потребления / А.А. Лазарев, М.В. Бейлин // Сборник научных трудов ХГПУ.– Х.: ХГПУ, 1999. – Вып. 74. – С. 27-29.

33. Ланецкий Б.Н. Основы теории надежности, технического обслуживания и ремонта вооружения и военной техники: Справочные материалы, часть 1. / Б.Н. Ланецкий, А.А. Посудевский. – Харьков: ХВУ, 1993. – 308 с.

34. Ланецкий Б.Н. Основы теории надежности, технического обслуживания и ремонта вооружения и военной техники: Справочные материалы, часть 2. / Б.Н. Ланецкий, А.А. Посудевский. – Харьков: ХВУ, 1993. – 208 с.

35. Лапсарь А.П. Метод оценки состояния сложных технических объектов для синтеза быстродействующих прогнозирующих систем / А.П. Лапсарь // Измерительная техника. – 2004. – № 2. – С. 7-10.

36. Линейные задачи оптимизации: Учеб. пособие / С.В. Лутманов. – Пермь: ЛИТЕР-А, 2004. – Ч.1. – Линейное программирование. – 128 с.

37. Литвак Б.Г. Экспертные технологии в управлении. Учебное пособие / Б.Г. Литвак. – М.: Дело, 2014. – 318 с.

38. Локазюк В.М. Надійність, контроль, діагностика і модернізація ПК: Посібн. / В.М. Локазюк, Ю.Г. Савченко. – К.: Видавничий центр «Академія», 2004. – 376 с.

39. Лопатников Л. И. Экономико-математический словарь: Словарь современной экономической науки / Л.И. Лопатников. – М.: Дело, 2003. – 520 с.

40. Манухина С.Ю. Инженерная психология и эргономика: хрестоматия / С.Ю Манухина. – М.: Изд. центр ЕАОИ, 2009. –224 с.

41. Мартыненко М.В. Человекомашинные процедуры поддержки организационно–управленческих решений: учеб. пособие СПбГЭТУ / М.В. Мартыненко, О.И. Шеховцов. – СПб, 2012. – 250 с.

42. Мунипов О.В. Эргономика: человекоориентированное проектирование техники, программных средств и среды: Учебник / О.В. Мунипов, В.П. Зинченко. – М.: Логос, 2001. – 356 с.

43. Надеев А.И. Математическая модель эксплуатационной надежности интеллектуальных датчиков / А.И. Надеев, Р.А. Юсупов,

Ю.К. Свечников, Д.Р. Юсупов // Измерительная техника. – М: Стандартинформ, 2004. – № 1. – С. 8-11.

44. Надійність техніки. Аналіз надійності. Основні положення: ДСТУ 2861-94 – [Чинний від 1997–01–01]. – Київ: Держстандарт України, 1995. – 33 с. – (Національний стандарт України).

45. Надійність техніки. Терміни та визначення: ДСТУ 2860-94 – [Чинний від 1996–01–01]. – Київ: Держстандарт України, 1994. – 36 с. – (Національний стандарт України).

46. Нейлор К. Как построить свою экспертную систему / К. Нейлор. – М.: Энегроатомиздат, 2007. – 242 с.

47. Николаева И. П. Экономический словарь / И.П. Николаева. – Проспект, 2015. – 399 с.

48. Онищук А.Г. Радиоприемные устройстваю: Учебн. пособ. – 2-е изд., испр. / А.Г. Онищук, И.И. Забеньков, А.М. Амелин. – Минск: Новое знание, 2007. – 240 с.

49. Осипов В. Базы данных и Delphi. Теория и практика / В. Осипов. – БХВ-Петербург, 2011. – 752 с.

50. Павленко М.А. Метод кольорового кодування інформаційних елементів при розробці інформаційних моделей в перспективних АСУ / М.А. Павленко, П.Г. Берднік, Д.В. Прибильнов // Наукова весна – 2008: Матеріали міжнародної наук. – практ. конф. – Х.: МСУ, 2008. – С. 25-27.

Додаток А
(обов'язковий)

Технічне завдання

Зміст

					ВКРБ-123.23.0008.00.00.ТЗ		
Вим.	Арк.	№ документа	Підпис	Дата			
Розробив		Семенюк О.В.			Програмне забезпечення системи тестування жорсткого диску з використанням технології SMART	Літ.	Аркуш 1 / Аркушів 6
Перевірив		Буравченко К.О.				Б	
Н. Контр.		Гермак В.С.				ЦНТУ КМ-19	
Затв.		Смірнов О.А.					

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку системи тестування жорсткого диску з використанням технології SMART.

2 Підстава для розробки

Підставою для розробки служить завдання на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, видане на кафедрі кібербезпеки та програмного забезпечення (нак. № 10-02 від 5.01.2023 року).

3 Мета та призначення розробки

Метою випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти є розробка програмного забезпечення системи тестування жорсткого диску з використанням технології SMART.

4 Джерела розробки

Джерелом цієї випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти є стосовна до теми література і існуючі аналоги.

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

– розробка програмної частин системи, а також розробка взаємодії системи з ОС та з користувачем;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– системи тестування жорсткого диску з використанням технології SMART;

– цілісність даних у процесі роботи та при зберіганні;

– простий, інтуїтивно зрозумілий інтерфейс.

5.3 Вимоги до функціональних характеристик

Розроблене програмне забезпечення не повинно мати обмежень на версію драйверів та операційної системи.

5.4 Вимоги до архітектури

Компонент, що розробляється повинен використовувати системні засоби та апаратні засоби, що на даному етапі розвитку обчислювальної техніки найбільше поширені.

5.5 Вимоги до надійності

Програмні модулі написані по всім правилам, які стосуються стандартних викликів процедур, функцій, методів і форм, визначених технічною документацією на середовище розробки.

5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

– температура повітря: 19-20 град. по Цельсію;

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

5.7 Вимоги до складу та параметрів технічних засобів

Програмне забезпечення повинно бути реалізоване на ПЕОМ архітектури IBM PC, працювати в ОС Windows 10/11 і з сумісними з цією платформою пристроями і прикладним програмним забезпеченням.

5.8 Вимоги до інформаційної і програмної сумісності

Переносність програмного забезпечення повинна бути забезпечена за рахунок його реалізації стандартного інтерфейсу взаємодії з ОС, що працюють під управлінням ОС Windows 10/11.

5.8.1 Обладнання

Комп'ютер Intel® Celeron/8 Mb/1.2 Gb/SVGA 14" 1Mb або сумісні з ним.

5.8.2 Мова програмування

Середовище Visual C++.

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

Програмна продукція повинна бути представлена у виді опису структури даних, схем та опису алгоритму, а також текстів вихідних модулів програмного забезпечення згідно ЄСПД .

7 Перелік документів, що розробляються

– Структурна схема системи – 1 аркуш.
– Функціональна схема системи – 1 аркуш.
– Діаграма процесів – 1 аркуш.
– Блок-схема алгоритму роботи програми – 2 аркуша.
– Пояснювальна записка – 70 аркушів.

8 Етапи розробки

8.1 Збір і обробка інформації по темі випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти. Постановка задачі на виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти (складання ТЗ).

8.2 Проведення досліджень або експериментальних робіт для уточнення основних положень випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти.

8.3 Розробка функціональних схем, блок схем алгоритмів роботи програмного забезпечення.

8.4 Побудова схем взаємодії даних.

8.5 Створення прототипу ПЗ.

8.6 Віднаходження ПЗ, аналіз отриманих результатів.

8.7 Оформлення пояснювальної записки і виконання робіт по графічній частині.

9 Порядок контролю та приймання

9.1 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на попередній захист 23.05.2023 р.

9.2 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на захист 8.06.2023 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

Керівник випускної кваліфікаційної роботи за

першим (бакалаврським) рівнем вищої освіти

_____ Буравченко К.О.

Програмне забезпечення системи тестування жорсткого диску з використанням технології SMART

Лістинг програми

Код документу 12

Носій: CD/DVD-диск / USB-флеш-накопичувач

Загальна кількість аркушів: 72

Літера: РП

Кропивницький – 2023 року

Файл DiskInfo.cpp – основна програма

```
#include "stdafx.h"
#include "DiskInfo.h"
#include "DiskInfoDlg.h"
#include "GraphDlg.h"

#include "GetFileVersion.h"
#include "GetOsInfo.h"
#include "IsCurrentUserLocalAdministrator.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// CDiskInfoApp

BEGIN_MESSAGE_MAP(CDiskInfoApp, CWinApp)
	ON_COMMAND(ID_HELP, &CWinApp::OnHelp)
END_MESSAGE_MAP()


// CDiskInfoApp конструктор

CDiskInfoApp::CDiskInfoApp()
{
	// ПРИМІТКА: Додаємо код конструктору,
	// Встановлюємо усі значиму ініціалізацію в  InitInstance
}


// Опис CDiskInfoApp об’єкту

CDiskInfoApp theApp;
CString m_Ini;

//-----------------------------------------------------------------------------
// Опис прототипів
//-----------------------------------------------------------------------------
static BOOL IsFileExistEx(const TCHAR* path, const TCHAR* fileName);
static BOOL RunAsRestart();
// CDiskInfoApp ініціалізація

BOOL CDiskInfoApp::InitInstance()
{
	BOOL flagEarthlight = FALSE;
	DWORD defaultDisk = 0;
	HANDLE hMutex = NULL;

	INITCOMMONCONTROLSEX InitCtrls;
	InitCtrls.dwSize = sizeof(InitCtrls);
	InitCtrls.dwICC = ICC_WIN95_CLASSES;
	InitCommonControlsEx(&InitCtrls);
	CWinApp::InitInstance();

	ZeroMemory(&m_OsVer, sizeof(OSVERSIONINFO));
	m_OsVer.dwOSVersionInfoSize = sizeof(OSVERSIONINFO);
	GetVersionEx(&m_OsVer);

	// Якщо версія підтримується.
	if(GetFileVersion(_T("Shdocvw.dll")) < 471)
	{
		AfxMessageBox(_T("CrystalDiskInfo має потребу у  IE 6.0 або
вище."));
	}

	// ініціалізація m_Ini
```

```
	TCHAR *ptrEnd;
	TCHAR ini[MAX_PATH];
	::GetModuleFileName(NULL, ini, MAX_PATH);
	if((ptrEnd = _tcsrchr(ini, '.')) != NULL)
	{
		*ptrEnd = '\0';
		_tcscat_s(ini, MAX_PATH, _T(".ini"));
	}
	m_Ini = ini;

	int argc;
	LPWSTR *argv = CommandLineToArgvW(GetCommandLineW(), &argc);

	if(argc > 1)
	{
		CString cstr;
		cstr = argv[1];
		if(cstr.Compare(_T("/Earthlight")) == 0)
		{
			flagEarthlight = TRUE;
			if(argc > 2)
			{
				defaultDisk = _tstoi(argv[2]);
			}
		}
		if(cstr.Compare(_T("/Startup")) == 0)
		{
			int time = 0;
			time = GetPrivateProfileInt(_T("Setting"),
_T("StartupWaitTime"), 30, m_Ini);
			if(time >= 0)
			{
				Sleep(time * 1000);
			}
			TCHAR str[MAX_PATH];
			::GetModuleFileName(NULL, str, MAX_PATH);
			ShellExecute(NULL, NULL, str, NULL, NULL, SW_SHOWNORMAL);
			return FALSE;
		}
	}

	// Розшифрування отриманих даних
	//flagEarthlight = TRUE;

	if(! flagEarthlight)
	{
		hMutex = ::CreateMutex(NULL, FALSE, PRODUCT_NAME PRODUCT_VERSION);
		if(GetLastError() == ERROR_ALREADY_EXISTS)
		{
			return FALSE;
		}
	}

	CString DefaultTheme;
	CString DefaultLanguage;
	TCHAR tmp[MAX_PATH];

	GetModuleFileName(NULL, tmp, MAX_PATH);
	if((ptrEnd = _tcsrchr(tmp, '\\')) != NULL)
	{
		*ptrEnd = '\0';
	}

	m_ExeDir.Format(_T("%s\\"), tmp);
	m_ThemeDir.Format(_T("%s\\%s"), tmp, THEME_DIR);
	m_LangDir.Format(_T("%s\\%s"), tmp, LANGUAGE_DIR);
	m_SmartDir.Format(_T("%s\\%s"), tmp, SMART_DIR);

	m_ThemeIndex = MENU_THEME_INDEX;
```

```
	m_LangIndex = MENU_LANG_INDEX;

	DefaultTheme.Format(_T("%s\\%s"), tmp, DEFAULT_THEME);
	DefaultLanguage.Format(_T("%s\\%s"), tmp, DEFAULT_LANGUAGE);

/*
	if(IsClassicSystem())
	{
		m_MainDlgPath.Format(_T("%s\\") DIALOG_DIR CLASSIC_DIALOG, tmp);
	}
*/
	if(! IsFileExist(m_MainDlgPath))
	{
		m_MainDlgPath.Format(_T("%s\\") DIALOG_DIR MAIN_DIALOG, tmp);
	}

	m_AboutDlgPath.Format(_T("%s\\") DIALOG_DIR ABOUT_DIALOG, tmp);
	m_SettingDlgPath.Format(_T("%s\\") DIALOG_DIR SETTING_DIALOG, tmp);
	if(GetIeVersion() == 800)
	{
		m_GraphDlgPath.Format(_T("%s\\") DIALOG_DIR GRAPH_DIALOG_IE8, tmp);
	}
	else
	{
		m_GraphDlgPath.Format(_T("%s\\") DIALOG_DIR GRAPH_DIALOG, tmp);
	}
	m_OptionDlgPath.Format(_T("%s\\") DIALOG_DIR OPTION_DIALOG, tmp);

	if(! IsFileExistEx(m_MainDlgPath, MAIN_DIALOG))
	{
		return FALSE;
	}
	if(! IsFileExistEx(m_AboutDlgPath, ABOUT_DIALOG))
	{
		return FALSE;
	}
	if(! IsFileExistEx(m_SettingDlgPath, SETTING_DIALOG))
	{
		return FALSE;
	}

// для сімейства Windows NT
#ifdef _UNICODE
	if(! IsCurrentUserLocalAdministrator())
	{
		if(m_OsVer.dwMajorVersion < 6)
		{
			AfxMessageBox(_T("CrystalDiskInfo is required Administrator
Privileges."));
		}
		RunAsRestart();
		return FALSE;
	}
#endif

	if(flagEarthlight)
	{
		CGraphDlg dlg(NULL, defaultDisk);
		m_pMainWnd = &dlg;
		INT_PTR nResponse = dlg.DoModal();
	}
	else
	{
		CDiskInfoDlg dlg;
		m_pMainWnd = &dlg;
		INT_PTR nResponse = dlg.DoModal();
		::ReleaseMutex(hMutex);
		::CloseHandle(hMutex);
	}
```

```
	return FALSE;
}

BOOL IsFileExistEx(const TCHAR* path, const TCHAR* fileName)
{
	if(! IsFileExist(path))
	{
		CString cstr;
		cstr.Format(_T("Не знайдено\"%s\"."), fileName);
		AfxMessageBox(cstr);
		return FALSE;
	}
	return TRUE;
}

BOOL RunAsRestart()
{
	int count;
#ifdef _UNICODE
	TCHAR** cmd = ::CommandLineToArgvW(::GetCommandLine(), &count);
#else
	TCHAR** cmd = ::__argv;
	count = ::__argc;
#endif

	if(count < 2 || _tcscmp(cmd[1], _T("runas")) != 0)
	{
		TCHAR path[MAX_PATH];
		::GetModuleFileName(NULL, path, MAX_PATH);
		if(::ShellExecute(NULL, _T("runas"), path, _T("runas"), NULL,
SW_SHOWNORMAL)
			> (HINSTANCE)32)
		{
			return TRUE;
		}
	}
	return FALSE;
}
```

Файл DiskInfo.h – бібліотека для файлу DiskInfo.cpp

```
#pragma once

#ifndef __AFXWIN_H__
	#error "include 'stdafx.h' перед включенням цього файлу для PCH"
#endif

#include "resource.h"		// головні символи

#define THEME_DIR                       _T("CdiResource\\theme\\")
#define LANGUAGE_DIR                    _T("CdiResource\\language\\")
#define DIALOG_DIR                            _T("CdiResource\\dialog\\")
#define SMART_DIR                       _T("Smart\\")
#define SMART_INI                       _T("Smart.ini")
#define EXCHANGE_INI                    _T("Exchange.ini")

#define MENU_THEME_INDEX                3
#define MENU_LANG_INDEX                 6
#define MENU_DRIVE_INDEX                4

#define MAIN_DIALOG                           _T("Main.html")
//#define CLASSIC_DIALOG                      _T("Classic.html")
#define ABOUT_DIALOG                    _T("About.html")
#define SETTING_DIALOG                  _T("Setting.html")
#define GRAPH_DIALOG                    _T("Graph.html")
#define GRAPH_DIALOG_IE8                _T("GraphIe8.html")
#define OPTION_DIALOG                   _T("Option.html")

#define DEFAULT_THEME                   THEME_DIR _T("default\\Main.css")
#define DEFAULT_LANGUAGE                LANGUAGE_DIR _T("English.lang")


class CDiskInfoApp : public CWinApp
{
public:
	CDiskInfoApp();

	OSVERSIONINFO m_OsVer;
	BOOL m_IsNT;

	CString m_MainDlgPath;
	CString m_AboutDlgPath;
	CString m_SettingDlgPath;
	CString m_GraphDlgPath;
	CString m_OptionDlgPath;
	CString m_SmartDir;
	CString m_ExeDir;
	CString m_Ini;

	CString m_ThemeDir;
	CString m_LangDir;
	DWORD m_ThemeIndex;
	DWORD m_LangIndex;

// Аннулюється
	public:
	virtual BOOL InitInstance();

// Реалізація

	DECLARE_MESSAGE_MAP()
};

extern CDiskInfoApp theApp;
```

Файл AtaSmart.cpp – тестування жорсткого диску з використанням технології SMART

```
#include "stdafx.h"
#include "AtaSmart.h"
#include <wbemcli.h>

#include "DebugPrint.h"
#include "DnpService.h"

#pragma comment(lib, "wbemuuid.lib")
#define SAFE_RELEASE(p) { if(p) { (p)->Release(); (p)=NULL; } }

static const TCHAR *commandTypeString[] =
{
	_T("pd"),
	_T("sm"),
	_T("sa"),
	_T("sp"),
	_T("io"),
	_T("lo"),
	_T("jm"),
	_T("cy")
};
// Визначення ATA
CAtaSmart::CAtaSmart()
{
	BOOL bosVersionInfoEx;
	ZeroMemory(&m_Os, sizeof(OSVERSIONINFOEX));
	m_Os.dwOSVersionInfoSize = sizeof(OSVERSIONINFOEX);
	if(!(bosVersionInfoEx = GetVersionEx((OSVERSIONINFO *)&m_Os)))
	{
		m_Os.dwOSVersionInfoSize = sizeof(OSVERSIONINFO);
		GetVersionEx((OSVERSIONINFO *)&m_Os);
	}

	m_FlagAtaPassThrough = FALSE;
	if(m_Os.dwMajorVersion >= 6 || (m_Os.dwMajorVersion == 5 &&
m_Os.dwMinorVersion == 2))
	{
		m_FlagAtaPassThrough = TRUE;
	}
	else if(m_Os.dwMajorVersion == 5 && m_Os.dwMinorVersion == 1)
	{
		CString cstr;
		cstr = m_Os.szCSDVersion;
		cstr.Replace(_T("Service Pack "), _T(""));
		if(_tstoi(cstr) >= 2)
		{
			m_FlagAtaPassThrough = TRUE;
		}
	}
}

CAtaSmart::~CAtaSmart()
{
}

/* ЗАГАЛЬНІ ФУНКЦІЇ*/
DWORD CAtaSmart::UpdateSmartInfo(DWORD i)
{
	if(vars.GetCount() == 0)
	{
		return SMART_STATUS_NO_CHANGE;
	}

	static SMART_ATTRIBUTE attribute[MAX_DISK][MAX_ATTRIBUTE] = {0};
// Читання таблиці атрибутів
```

```
	if(vars[i].IsSmartEnabled)
	{
		switch(vars[i].CommandType)
		{
		case CMD_TYPE_PHYSICAL_DRIVE:
			GetSmartAttributePd(vars[i].PhysicalDriveId, &(vars[i]));
			vars[i].DiskStatus = CheckDiskStatus(vars[i].Attribute,
vars[i].Threshold, vars[i].AttributeCount, vars[i].VendorId);
			break;

		case CMD_TYPE_SCSI_MINIPORT:
			GetSmartAttributeScsi(vars[i].ScsiPort, vars[i].ScsiTargetId,
&(vars[i]));
			vars[i].DiskStatus = CheckDiskStatus(vars[i].Attribute,
vars[i].Threshold, vars[i].AttributeCount, vars[i].VendorId);
			break;
		case CMD_TYPE_SAT:
		case CMD_TYPE_SUNPLUS:
		case CMD_TYPE_IO_DATA:
		case CMD_TYPE_LOGITEC:
		case CMD_TYPE_JMICRON:
		case CMD_TYPE_CYPRESS:
			GetSmartAttributeSat(vars[i].PhysicalDriveId, &(vars[i]));
			vars[i].DiskStatus = CheckDiskStatus(vars[i].Attribute,
vars[i].Threshold, vars[i].AttributeCount, vars[i].VendorId);
			break;
		default:
			return SMART_STATUS_NO_CHANGE;
		}

		return CheckSmartAttributeUpdate(i, attribute[i],
vars[i].Attribute);
	}

	return SMART_STATUS_NO_CHANGE;
}

/* ЗАГАЛЬНІ ФУНКЦІЇ*/
BOOL CAtaSmart::UpdateIdInfo(DWORD i)
{
	BOOL flag = FALSE;
	switch(vars[i].CommandType)
	{
	case CMD_TYPE_PHYSICAL_DRIVE:
		flag =  DoIdentifyDevicePd(vars[i].PhysicalDriveId,
&(vars[i].IdentifyDevice));
		break;
	case CMD_TYPE_SCSI_MINIPORT:
		flag =  DoIdentifyDeviceScsi(vars[i].ScsiPort, vars[i].ScsiTargetId,
&(vars[i].IdentifyDevice));
		break;
	case CMD_TYPE_SAT:
	case CMD_TYPE_SUNPLUS:
	case CMD_TYPE_IO_DATA:
	case CMD_TYPE_LOGITEC:
	case CMD_TYPE_JMICRON:
	case CMD_TYPE_CYPRESS:
		flag = DoIdentifyDeviceSat(vars[i].PhysicalDriveId,
&(vars[i].IdentifyDevice), vars[i].CommandType);
		break;
	default:
		return FALSE;
		break;
	}

	if(vars[i].Major >= 3 && vars[i].IdentifyDevice.CommandSetSupported2 & (1
<< 3))
	{
		vars[i].IsApmSupported = TRUE;
```

```
            if(vars[i].IdentifyDevice.CommandSetEnabled2 & (1 << 3))
            {
                  vars[i].IsApmEnabled = TRUE;
            }
            else
            {
                  vars[i].IsApmEnabled = FALSE;
            }
      }
      if(vars[i].Major >= 5 && vars[i].IdentifyDevice.CommandSetSupported2 & (1
<< 9))
      {
            vars[i].IsAamSupported = TRUE;
            if(vars[i].IdentifyDevice.CommandSetEnabled2 & (1 << 9))
            {
                  vars[i].IsAamEnabled = TRUE;
            }
            else
            {
                  vars[i].IsAamEnabled = FALSE;
            }
      }

      return flag;
}

/* ЗАГАЛЬНІ ФУНКЦІЇ*/
BYTE CAtaSmart::GetAamValue(DWORD i)
{
      return LOBYTE(vars[i].IdentifyDevice.AcoustricManagement);
}

/* ЗАГАЛЬНІ ФУНКЦІЇ*/
BYTE CAtaSmart::GetApmValue(DWORD i)
{
      return LOBYTE(vars[i].IdentifyDevice.CurrentPowerManagement);
}

/* ЗАГАЛЬНІ ФУНКЦІЇ*/
BYTE CAtaSmart::GetRecommendAamValue(DWORD i)
{
      return HIBYTE(vars[i].IdentifyDevice.AcoustricManagement);
}

/* ЗАГАЛЬНІ ФУНКЦІЇ*/
BYTE CAtaSmart::GetRecommendApmValue(DWORD i)
{
      return HIBYTE(vars[i].IdentifyDevice.CurrentPowerManagement);
}

/* ЗАГАЛЬНІ ФУНКЦІЇ*/
BOOL CAtaSmart::EnableAam(DWORD i, BYTE param)
{
      return SendAtaCommand(i, 0xEF, 0x42, param);
}

/* ЗАГАЛЬНІ ФУНКЦІЇ*/
BOOL CAtaSmart::DisableAam(DWORD i)
{
      return SendAtaCommand(i, 0xEF, 0xC2, 0);
}

/* ЗАГАЛЬНІ ФУНКЦІЇ*/
BOOL CAtaSmart::EnableApm(DWORD i, BYTE param)
{
      return SendAtaCommand(i, 0xEF, 0x05, param);
}

/* ЗАГАЛЬНІ ФУНКЦІЇ*/
```

```
BOOL CAtaSmart::DisableApm(DWORD i)
{
	return SendAtaCommand(i, 0xEF, 0x85, 0);
}

BOOL CAtaSmart::SendAtaCommand(DWORD i, BYTE main, BYTE sub, BYTE param)
{
	switch(vars[i].CommandType)
	{
	case CMD_TYPE_PHYSICAL_DRIVE:
		return SendAtaCommandPd(vars[i].PhysicalDriveId, main, sub, param,
NULL, 0);
		break;
	case CMD_TYPE_SCSI_MINIPORT:
		return SendAtaCommandScsi(vars[i].ScsiPort, vars[i].ScsiTargetId,
main, sub, param);
		break;
	case CMD_TYPE_SAT:
	case CMD_TYPE_SUNPLUS:
	case CMD_TYPE_IO_DATA:
	case CMD_TYPE_LOGITEC:
	case CMD_TYPE_JMICRON:
	case CMD_TYPE_CYPRESS:
		return SendAtaCommandSat(vars[i].PhysicalDriveId, main, sub, param,
vars[i].CommandType);
		break;
	default:
		return FALSE;
		break;
	}
	return FALSE;
}

DWORD CAtaSmart::CheckSmartAttributeUpdate(DWORD index, SMART_ATTRIBUTE* pre,
SMART_ATTRIBUTE* cur)
{

	if(memcmp(pre, cur, sizeof(SMART_ATTRIBUTE) * MAX_ATTRIBUTE) == 0)
	{
		return SMART_STATUS_NO_CHANGE;
	}
	else
	{
		for(int i = 0; i < MAX_ATTRIBUTE; i++)
		{
			switch(cur[i].Id)
			{
			case 0x09: // Кількість відпрацьованих годин у включеному
стані
				{
					DWORD preRawValue = MAKELONG(
						MAKEWORD(pre[i].RawValue[0],
pre[i].RawValue[1]),
						MAKEWORD(pre[i].RawValue[2],
pre[i].RawValue[3])
						);
					DWORD curRawValue = MAKELONG(
						MAKEWORD(cur[i].RawValue[0],
cur[i].RawValue[1]),
						MAKEWORD(cur[i].RawValue[2],
cur[i].RawValue[3])
						);

					if(GetPowerOnHours(preRawValue,
vars[index].DetectedTimeUnitType)
					!= GetPowerOnHours(curRawValue,
vars[index].DetectedTimeUnitType))
					{
```

```
                                    memcpy(pre, cur, sizeof(SMART_ATTRIBUTE) *
MAX_ATTRIBUTE);
                                    return SMART_STATUS_MAJOR_CHANGE;
                              }
                              if(GetPowerOnHours(preRawValue,
vars[index].MeasuredTimeUnitType)
                              != GetPowerOnHours(curRawValue,
vars[index].MeasuredTimeUnitType))
                              {
                                    memcpy(pre, cur, sizeof(SMART_ATTRIBUTE) *
MAX_ATTRIBUTE);
                                    return SMART_STATUS_MAJOR_CHANGE;
                              }
                        }
                        break;
                  case 0x0C: // Кількість зафіксованих повторів
включення/вимикання живлення накопичувача
                        {
                              DWORD preRawValue = MAKELONG(
                                    MAKEWORD(pre[i].RawValue[0],
pre[i].RawValue[1]),
                                    MAKEWORD(pre[i].RawValue[2],
pre[i].RawValue[3])
                                    );
                              DWORD curRawValue = MAKELONG(
                                    MAKEWORD(cur[i].RawValue[0],
cur[i].RawValue[1]),
                                    MAKEWORD(cur[i].RawValue[2],
cur[i].RawValue[3])
                                    );
                              if(preRawValue != curRawValue)
                              {
                                    memcpy(pre, cur, sizeof(SMART_ATTRIBUTE) *
MAX_ATTRIBUTE);
                                    return SMART_STATUS_MAJOR_CHANGE;
                              }
                        }
                        break;
                  case 0xC2: // Температура
                        if(pre[i].RawValue[0] != cur[i].RawValue[0]
                        || pre[i].CurrentValue != cur[i].CurrentValue)
                        {
                              memcpy(pre, cur, sizeof(SMART_ATTRIBUTE) *
MAX_ATTRIBUTE);
                              return SMART_STATUS_MAJOR_CHANGE;
                        }
                        break;
                  default:
                        break;
                  }
            }
            return SMART_STATUS_MINOR_CHANGE;
      }
}

/* ЗАГАЛЬНІ ФУНКЦІЇ*/
BOOL CAtaSmart::MeasuredTimeUnit()
{
      DWORD getTickCount = GetTickCount();
      if(getTickCount > MeasuredGetTickCount + 155000 || MeasuredGetTickCount +
125000 > getTickCount)
      {
            return FALSE;
      }

      for(int i = 0; i < vars.GetCount(); i++)
      {
            if(vars[i].PowerOnRawValue < 0)
            {
```

```
                continue;
            }
            UpdateSmartInfo(i);

            DWORD test = vars[i].PowerOnRawValue - vars[i].PowerOnStartRawValue;

            if(vars[i].Model.Find(_T("SAMSUNG")) == 0)
            {
                if(test >= 2)
                {
                    vars[i].MeasuredTimeUnitType = POWER_ON_HALF_MINUTES;
                }
                else
                {
                    vars[i].MeasuredTimeUnitType = POWER_ON_HOURS;
                }
            }
            else if(vars[i].Model.Find(_T("FUJITSU")) == 0)
            {
                if(test >= 2)
                {
                    vars[i].MeasuredTimeUnitType = POWER_ON_SECONDS;
                }
                else
                {
                    vars[i].MeasuredTimeUnitType = POWER_ON_HOURS;
                }
            }
            else if(vars[i].Model.Find(_T("MAXTOR")) == 0)
            {
                if(test >= 2)
                {
                    vars[i].MeasuredTimeUnitType = POWER_ON_MINUTES;
                    vars[i].IsMaxtorMinute = TRUE;
                }
                else
                {
                    vars[i].MeasuredTimeUnitType = POWER_ON_HOURS;
                    vars[i].IsMaxtorMinute = FALSE;
                }
            }
            else
            {
                if(test >= 2)
                {
                    vars[i].MeasuredTimeUnitType = POWER_ON_MINUTES;
                }
                else
                {
                    vars[i].MeasuredTimeUnitType = POWER_ON_HOURS;
                }
            }
        }

    return TRUE;
}

/* ЗАГАЛЬНІ ФУНКЦІЇ*/
BOOL CAtaSmart::Init(BOOL useWmi, BOOL advancedDiskSearch, PBOOL flagChangeDisk)
{
    IsEnabledWmi = FALSE;
    CArray<DISK_POSITION, DISK_POSITION> previous;

    if(flagChangeDisk != NULL)
    {
        *flagChangeDisk = FALSE;
        for(int i = 0; i < vars.GetCount(); i++)
        {
            DISK_POSITION dp;
```

```
                dp.PhysicalDriveId = vars.GetAt(i).PhysicalDriveId;
                dp.ScsiTargetId = vars.GetAt(i).ScsiTargetId;
                dp.ScsiPort = vars.GetAt(i).ScsiPort;

                previous.Add(dp);
            }
        }

        // Ініціалізація
        vars.RemoveAll();
        m_ControllerMap = _T("");

        if(useWmi)
        {
            HRESULT hRes = S_OK;
            ULONG   uReturned = 1;

            IWbemLocator*                pIWbemLocator = NULL;
            IWbemServices*               pIWbemServices = NULL;
            IEnumWbemClassObject*   pEnumCOMDevs = NULL;
            IEnumWbemClassObject*   pEnumCOMDevs2 = NULL;
            IWbemClassObject*       pCOMDev = NULL;

            DebugPrint(_T("CAtaSmart::Init WMI on - Start"));

            bool initWmi = true;
            CDnpService cService;

            for(int i = 0; i < 3; i++)
            {
                if(! cService.IsServiceRunning(_T("Winmgmt")))
                {
                    DebugPrint(_T("Waiting... Winmgmt"));
                    initWmi = cService.EasyStart(_T("Winmgmt"));
                    continue;
                }
                else
                {
                    break;
                }
            }

            if(initWmi)
            {
                try
                {
                    DebugPrint(_T("CoInitialize()"));
                    CoInitialize(NULL);
                    DebugPrint(_T("CoInitializeSecurity()"));
                    CoInitializeSecurity(NULL, -1, NULL, NULL,
RPC_C_AUTHN_LEVEL_DEFAULT,
                        RPC_C_IMP_LEVEL_IMPERSONATE, NULL, EOAC_NONE,
NULL);
                    DebugPrint(_T("CoCreateInstance()"));
                    if(FAILED(CoCreateInstance(CLSID_WbemLocator, NULL,
CLSCTX_INPROC_SERVER,
                        IID_IWbemLocator, (LPVOID *)&pIWbemLocator)))
                    {
                        CoUninitialize();
                        DebugPrint(_T("NG:WMI Init 1"));
                    }
                    else
                    {
                        long securityFlag = 0;
                        if( m_Os.dwMajorVersion >= 6 // Vista або пізніша
версія
                        || (m_Os.dwMajorVersion == 5 &&
m_Os.dwMinorVersion >= 1) // XP або пізніша версія
                        )
```

```
                                {
                                      securityFlag =
WBEM_FLAG_CONNECT_USE_MAX_WAIT;
                                }

                                DebugPrint(_T("ConnectServer()"));
                                if(FAILED(pIWbemLocator-
>ConnectServer(SysAllocString(L"\\\\.\\root\\cimv2"),
                                      NULL, NULL, 0L,
                                      securityFlag,
                                      NULL, NULL, &pIWbemServices)))
                                {
                                      CoUninitialize();
                                      DebugPrint(_T("NG:WMI Init 2"));
                                }
                                else
                                {
                                      DebugPrint(_T("CoSetProxyBlanket()"));
                                      hRes = CoSetProxyBlanket(pIWbemServices,
RPC_C_AUTHN_WINNT, RPC_C_AUTHZ_NONE,
                                            NULL, RPC_C_AUTHN_LEVEL_CALL,
RPC_C_IMP_LEVEL_IMPERSONATE, NULL, EOAC_NONE);
                                      if(FAILED(hRes))
                                      {
                                            CoUninitialize();
                                            CString cstr;
                                            cstr.Format(_T("NG:WMI Init - %08X"),
hRes);
                                            DebugPrint(cstr);
                                      }
                                      else
                                      {
                                            IsEnabledWmi = TRUE;
                                            DebugPrint(_T("OK:WMI Init"));
                                      }
                                }
                                SAFE_RELEASE(pIWbemLocator);
                          }
                    }
                    catch(...)
                    {
                          DebugPrint(_T("EX:WMI Init"));
                    }
              }
              else
              {
                    DebugPrint(_T("NG:WMI Init 3"));
              }

              if(IsEnabledWmi)
              {
                    CStringArray csa;
                    CString temp, cstr, cstr1, cstr2;
                    try
                    {// Win32_IDE Контролер
                          pIWbemServices->ExecQuery(SysAllocString(L"WQL"),
                                SysAllocString(L"select Name, DeviceID from
Win32_IDEController"), WBEM_FLAG_FORWARD_ONLY | WBEM_FLAG_RETURN_IMMEDIATELY,
NULL, &pEnumCOMDevs);
                          while(pEnumCOMDevs && SUCCEEDED(pEnumCOMDevs-
>Next(10000, 1, &pCOMDev, &uReturned)) && uReturned == 1)
                          {
                                VARIANT  pVal;
                                VariantClear(&pVal);
                                CString name1, deviceId, channel;
                                if(pCOMDev->Get(L"DeviceID", 0L, &pVal, NULL,
NULL) == WBEM_S_NO_ERROR && pVal.vt > VT_NULL)
                                {
                                      deviceId = pVal.bstrVal;
```

```
                                    if(deviceId.Find(_T("PCIIDE\\IDECHANNEL"))
== 0)
                                    {
                                          channel = deviceId.Right(1);
                                    }
                                    deviceId.Replace(_T("\\"), _T("\\\\"));
                                    VariantClear(&pVal);
                              }
                              if(pCOMDev->Get(L"Name", 0L, &pVal, NULL, NULL) ==
WBEM_S_NO_ERROR && pVal.vt > VT_NULL)
                              {
                                    name1 = pVal.bstrVal;
                                    if(! channel.IsEmpty())
                                    {
                                          name1 += _T(" (") + channel + _T(")");
                                    }
                                    m_IdeController.Add(name1);

                                    VariantClear(&pVal);
                              }
                              SAFE_RELEASE(pCOMDev);

                              if(cstr.Find(name1) == -1 || cstr.Find(name1 +
_T(" [ATA]")) >= 0)
                              {
                                    csa.Add(cstr);
                                    cstr = _T("%%%") + name1 + _T(" [ATA]");
                                    cstr += _T("\r\n");
                              }

                              CString mapping;
                              mapping.Format(_T("ASSOCIATORS OF
{Win32_IDEController.DeviceID=\"%s\"} WHERE AssocClass =
Win32_IDEControllerDevice"), deviceId);
                              pIWbemServices->ExecQuery(SysAllocString(L"WQL"),
SysAllocString(mapping), WBEM_FLAG_FORWARD_ONLY | WBEM_FLAG_RETURN_IMMEDIATELY,
NULL, &pEnumCOMDevs2);
                              while(pEnumCOMDevs2 && SUCCEEDED(pEnumCOMDevs2-
>Next(10000, 1, &pCOMDev, &uReturned)) && uReturned == 1)
                              {
                                    VARIANT pVal;
                                    VariantClear(&pVal);
                                    CString name2, deviceId, channel;
                                    if(pCOMDev->Get(L"DeviceID", 0L, &pVal,
NULL, NULL) == WBEM_S_NO_ERROR && pVal.vt > VT_NULL)
                                    {
                                          deviceId = pVal.bstrVal;

      if(deviceId.Find(_T("PCIIDE\\IDECHANNEL")) == 0)
                                          {
                                                channel = deviceId.Right(1);
                                          }
                                          VariantClear(&pVal);
                                    }
                                    if(pCOMDev->Get(L"Name", 0L, &pVal, NULL,
NULL) == WBEM_S_NO_ERROR && pVal.vt > VT_NULL)
                                    {
                                          name2 = pVal.bstrVal;
                                          if(! channel.IsEmpty())
                                          {
                                                name2 += _T(" (") + channel +
_T(")");
                                          }
                                          VariantClear(&pVal);
                                    }
                                    SAFE_RELEASE(pCOMDev);

                                    if(cstr.Find(_T("   - ") + name1) >= 0 ||
cstr.Find(_T("   + ") + name1) >= 0)
```

```
                                    {
                                          cstr1 = _T("- ") + name1;
                                          cstr2 = _T("+ ") + name1;
                                          cstr.Replace(cstr1, cstr2);

                                          cstr1 = name1 + _T("\r\n     - ") +
name2;
                                          cstr.Replace(name1, cstr1);
                                    }
                                    else
                                    {
                                          cstr += _T("   - ") + name2 +
_T("\r\n");
                                    }
                                    cstr.Replace(_T("%%%"), _T(" + "));
                              }
                              cstr.Replace(_T("%%%"), _T(" - "));
                              SAFE_RELEASE(pEnumCOMDevs2);
                        }
                        csa.Add(cstr);
                        SAFE_RELEASE(pEnumCOMDevs);
                        DebugPrint(_T("OK:Win32_IDEController"));
                  }
                  catch(...)
                  {
                        DebugPrint(_T("EX:Win32_IDEController"));
                  }

                  try
                  {
                        cstr = _T("");
                        pIWbemServices->ExecQuery(SysAllocString(L"WQL"),
                              SysAllocString(L"select Name, DeviceID from
Win32_SCSIController"), WBEM_FLAG_FORWARD_ONLY | WBEM_FLAG_RETURN_IMMEDIATELY,
NULL, &pEnumCOMDevs);
                        while(pEnumCOMDevs && SUCCEEDED(pEnumCOMDevs-
>Next(10000, 1, &pCOMDev, &uReturned)) && uReturned == 1)
                        {
                              VARIANT  pVal;
                              VariantClear(&pVal);
                              CString name1, deviceId;
                              if(pCOMDev->Get(L"DeviceID", 0L, &pVal, NULL,
NULL) == WBEM_S_NO_ERROR && pVal.vt > VT_NULL)
                              {
                                    deviceId = pVal.bstrVal;
                                    deviceId.Replace(_T("\\"), _T("\\\\"));
                                    VariantClear(&pVal);
                              }
                              if(pCOMDev->Get(L"Name", 0L, &pVal, NULL, NULL) ==
WBEM_S_NO_ERROR && pVal.vt > VT_NULL)
                              {
                                    name1 = pVal.bstrVal;
                                    m_ScsiController.Add(name1);
                                    VariantClear(&pVal);
                              }
                              SAFE_RELEASE(pCOMDev);
                              if(cstr.Find(name1) == -1 || cstr.Find(name1 +
_T(" [SCSI]")) >= 0)
                              {
                                    csa.Add(cstr);
                                    cstr = _T("%%%") + name1 + _T(" [SCSI]");

                                    cstr += _T("\r\n");
                              }

                              CString mapping;
                              mapping.Format(_T("ASSOCIATORS OF
{Win32_SCSIController.DeviceID=\"%s\"} WHERE AssocClass =
Win32_SCSIControllerDevice"), deviceId);
```

```
                              pIWbemServices->ExecQuery(SysAllocString(L"WQL"),
SysAllocString(mapping), WBEM_FLAG_FORWARD_ONLY | WBEM_FLAG_RETURN_IMMEDIATELY,
NULL, &pEnumCOMDevs2);
                              while(pEnumCOMDevs2 && SUCCEEDED(pEnumCOMDevs2-
>Next(10000, 1, &pCOMDev, &uReturned)) && uReturned == 1)
                              {
                                    VARIANT pVal;
                                    VariantClear(&pVal);
                                    CString name2, deviceId;
                                    if(pCOMDev->Get(L"DeviceID", 0L, &pVal,
NULL, NULL) == WBEM_S_NO_ERROR && pVal.vt > VT_NULL)
                                    {
                                          deviceId = pVal.bstrVal;
                                          VariantClear(&pVal);
                                    }
                                    if(pCOMDev->Get(L"Name", 0L, &pVal, NULL,
NULL) == WBEM_S_NO_ERROR && pVal.vt > VT_NULL)
                                    {
                                          name2 = pVal.bstrVal;
                                          VariantClear(&pVal);
                                    }
                                    SAFE_RELEASE(pCOMDev);

                                    if(cstr.Find(_T("   - ") + name1) >= 0 ||
cstr.Find(_T("   + ") + name1) >= 0)
                                    {
                                          cstr1 = _T("- ") + name1;
                                          cstr2 = _T("+ ") + name1;
                                          cstr.Replace(cstr1, cstr2);

                                          cstr1 = name1 + _T("\r\n     - ") +
name2;
                                          cstr.Replace(name1, cstr1);
                                    }
                                    else
                                    {
                                          cstr += _T("   - ") + name2 +
_T("\r\n");
                                    }
                                    cstr.Replace(_T("%%%"), _T(" + "));
                              }
                              cstr.Replace(_T("%%%"), _T(" - "));
                              SAFE_RELEASE(pEnumCOMDevs2);
                        }
                        csa.Add(cstr);
                        SAFE_RELEASE(pEnumCOMDevs);
                        DebugPrint(_T("OK:Win32_SCSIController"));
                  }
                  catch(...)
                  {
                        DebugPrint(_T("EX:Win32_SCSIController"));
                  }

                  for(int i = 0; i < csa.GetCount(); i++)
                  {
                        m_ControllerMap += csa.GetAt(i);
                  }

                  try
                  {// Win32_USB Контролер
                        pIWbemServices->ExecQuery(SysAllocString(L"WQL"),
                              SysAllocString(L"select Name, DeviceID from
Win32_USBController"), WBEM_FLAG_FORWARD_ONLY | WBEM_FLAG_RETURN_IMMEDIATELY,
NULL, &pEnumCOMDevs);
                        while(pEnumCOMDevs && SUCCEEDED(pEnumCOMDevs-
>Next(10000, 1, &pCOMDev, &uReturned)) && uReturned == 1)
                        {
                              VARIANT  pVal;
                              VariantClear(&pVal);
```

```
                                CString deviceId, channel;
                                if(pCOMDev->Get(L"DeviceID", 0L, &pVal, NULL,
NULL) == WBEM_S_NO_ERROR && pVal.vt > VT_NULL)
                                {
                                      deviceId = pVal.bstrVal;
                                      deviceId.Replace(_T("\\"), _T("\\\\"));
                                      VariantClear(&pVal);
                                }
                                SAFE_RELEASE(pCOMDev);

                                CString mapping, enclosure;
                                mapping.Format(_T("ASSOCIATORS OF
{Win32_USBController.DeviceID=\"%s\"} WHERE AssocClass =
Win32_USBControllerDevice"), deviceId);
                                pIWbemServices->ExecQuery(SysAllocString(L"WQL"),
SysAllocString(mapping), WBEM_FLAG_FORWARD_ONLY | WBEM_FLAG_RETURN_IMMEDIATELY,
NULL, &pEnumCOMDevs2);
                                while(pEnumCOMDevs2 && SUCCEEDED(pEnumCOMDevs2-
>Next(10000, 1, &pCOMDev, &uReturned)) && uReturned == 1)
                                {
                                      VARIANT pVal;
                                      VariantClear(&pVal);
                                      if(pCOMDev->Get(L"DeviceID", 0L, &pVal,
NULL, NULL) == WBEM_S_NO_ERROR && pVal.vt > VT_NULL)
                                      {
                                            cstr = pVal.bstrVal;
                                            VariantClear(&pVal);
                                            if(cstr.Find(_T("USBSTOR")) >= 0)
                                            {
                                                  EXTERNAL_DISK_INFO edi = {0};
                                                  int curPos= 0;
                                                  CString resToken;
                                                  resToken =
deviceId.Tokenize(_T("\\&"), curPos);
                                                  while(resToken != _T(""))
                                                  {

      if(resToken.Replace(_T("VID_"), _T("")) > 0)
                                                        {
                                                              edi.VendorId =
_tcstol(resToken, NULL, 16);
                                                        }
                                                        else
if(resToken.Replace(_T("PID_"), _T("")) > 0)
                                                        {
                                                              edi.ProductId =
_tcstol(resToken, NULL, 16);
                                                        }
                                                        resToken =
deviceId.Tokenize(_T("\\&"), curPos);
                                                  };

                                                  if(pCOMDev->Get(L"Name", 0L,
&pVal, NULL, NULL) == WBEM_S_NO_ERROR && pVal.vt > VT_NULL)
                                                  {
                                                        edi.Enclosure =
pVal.bstrVal;
                                                        VariantClear(&pVal);
                                                  }
                                                  externals.Add(edi);
                                            }
                                            deviceId = cstr;
                                      }
                                      SAFE_RELEASE(pCOMDev);
                                }
                                SAFE_RELEASE(pEnumCOMDevs2);
                          }
                          SAFE_RELEASE(pEnumCOMDevs);
                          DebugPrint(_T("OK:Win32_USBController"));
```

```
		}
		catch(...)
		{
			DebugPrint(_T("EX:Win32_USBController"));
		}

		try
		{// Win32_1394 Контролер
			pIWbemServices->ExecQuery(SysAllocString(L"WQL"),
				SysAllocString(L"select Name, DeviceID from
Win32_1394Controller"), WBEM_FLAG_FORWARD_ONLY | WBEM_FLAG_RETURN_IMMEDIATELY,
NULL, &pEnumCOMDevs);
			while(pEnumCOMDevs && SUCCEEDED(pEnumCOMDevs-
>Next(10000, 1, &pCOMDev, &uReturned)) && uReturned == 1)
			{
				VARIANT  pVal;
				VariantClear(&pVal);
				CString deviceId, channel;
				if(pCOMDev->Get(L"DeviceID", 0L, &pVal, NULL,
NULL) == WBEM_S_NO_ERROR && pVal.vt > VT_NULL)
				{
					deviceId = pVal.bstrVal;
					deviceId.Replace(_T("\\"), _T("\\\\"));
					VariantClear(&pVal);
				}
				SAFE_RELEASE(pCOMDev);

				CString mapping, enclosure;
				mapping.Format(_T("ASSOCIATORS OF
{Win32_1394Controller.DeviceID=\"%s\"} WHERE AssocClass =
Win32_1394ControllerDevice"), deviceId);
				pIWbemServices->ExecQuery(SysAllocString(L"WQL"),
SysAllocString(mapping), WBEM_FLAG_FORWARD_ONLY | WBEM_FLAG_RETURN_IMMEDIATELY,
NULL, &pEnumCOMDevs2);
				while(pEnumCOMDevs2 && SUCCEEDED(pEnumCOMDevs2-
>Next(10000, 1, &pCOMDev, &uReturned)) && uReturned == 1)
				{
					VARIANT pVal;
					VariantClear(&pVal);
					if(pCOMDev->Get(L"DeviceID", 0L, &pVal,
NULL, NULL) == WBEM_S_NO_ERROR && pVal.vt > VT_NULL)
					{
						deviceId = pVal.bstrVal;
						VariantClear(&pVal);
					}
					SAFE_RELEASE(pCOMDev);
				}
				SAFE_RELEASE(pEnumCOMDevs2);
			}
			SAFE_RELEASE(pEnumCOMDevs);
			DebugPrint(_T("OK:Win32_1394Controller"));
		}
		catch(...)
		{
			DebugPrint(_T("EX:Win32_1394Controller"));
		}

		/* РОЗШИФРУВАННЯ ЗНАЧЕНЬ
		for(int i = 0; i < externals.GetCount(); i++)
		{
			CString cstr;
			cstr.Format(_T("Enclosure=%s, VID=%04X, PID=%04X"),
				externals.GetAt(i).Enclosure,
				externals.GetAt(i).VendorId,
				externals.GetAt(i).ProductId);

			AfxMessageBox(cstr);
		}
		*/
```

```
                  try
                  {
                        pIWbemServices->ExecQuery(SysAllocString(L"WQL"),
                              SysAllocString(L"SELECT * FROM Win32_DiskDrive"),
WBEM_FLAG_FORWARD_ONLY | WBEM_FLAG_RETURN_IMMEDIATELY, NULL, &pEnumCOMDevs);
                        DebugPrint(_T("DO:SELECT * FROM Win32_DiskDrive"));
                        while(pEnumCOMDevs && SUCCEEDED(pEnumCOMDevs-
>Next(10000, 1, &pCOMDev, &uReturned)) && uReturned == 1)
                        {
                              CString mapping1, mapping2;
                              CString model, deviceId, diskSize;
                              INT physicalDriveId = -1, scsiPort = -1,
scsiTargetId = -1;

                              VARIANT pVal;
                              VariantClear(&pVal);
                              if(pCOMDev->Get(L"Size", 0L, &pVal, NULL, NULL) ==
WBEM_S_NO_ERROR && pVal.vt > VT_NULL)
                              {
                                    diskSize = pVal.bstrVal;
                                    VariantClear(&pVal);
                              }
                              if(pCOMDev->Get(L"DeviceID", 0L, &pVal, NULL,
NULL) == WBEM_S_NO_ERROR && pVal.vt > VT_NULL)
                              {
                                    deviceId = pVal.bstrVal;
                                    deviceId.Replace(_T("\\"), _T("\\\\"));
                                    if(_ttoi(deviceId.Right(2)) >= 10)
                                    {
                                          physicalDriveId =
_ttoi(deviceId.Right(2));
                                    }
                                    else
                                    {
                                          physicalDriveId =
_ttoi(deviceId.Right(1));
                                    }
                                    VariantClear(&pVal);
                              }
                              if(pCOMDev->Get(L"Model", 0L, &pVal, NULL, NULL)
== WBEM_S_NO_ERROR && pVal.vt > VT_NULL)
                              {
                                    model = pVal.bstrVal;
                                    VariantClear(&pVal);
                              }
                              if(pCOMDev->Get(L"SCSIPort", 0L, &pVal, NULL,
NULL) == WBEM_S_NO_ERROR && pVal.vt > VT_NULL)
                              {
                                    scsiPort = pVal.intVal;
                                    VariantClear(&pVal);
                              }
                              if(pCOMDev->Get(L"SCSITargetId", 0L, &pVal, NULL,
NULL) == WBEM_S_NO_ERROR && pVal.vt > VT_NULL)
                              {
                                    scsiTargetId = pVal.intVal;
                                    VariantClear(&pVal);
                              }
                              SAFE_RELEASE(pCOMDev);

                              try
                              {
                                    // GetDiskInfo - інформація про диск
                                    CString cstr;
                                    cstr.Format(_T("DO:GetDiskInfo pd=%d, sp=%d,
st=%d"), physicalDriveId, scsiPort, scsiTargetId);
                                    DebugPrint(cstr);
```

```
                                    INTERFACE_TYPE interfaceType =
INTERFACE_TYPE_UNKNOWN;
                                    VENDOR_ID vendorId = VENDOR_UNKNOWN;

                                    if(model.Find(_T(" USB Device")) >= 0)
                                    {
                                          interfaceType = INTERFACE_TYPE_USB;
                                    }
                                    else if(model.Find(_T(" IEEE 1394 SBP2
Device")) >= 0)
                                    {
                                          interfaceType =
INTERFACE_TYPE_IEEE1394;
                                    }

                                    for(int i = 0; i < externals.GetCount();
i++)
                                    {

      if(model.Find(externals.GetAt(i).Enclosure) == 0)
                                          {
                                                vendorId =
(VENDOR_ID)externals.GetAt(i).VendorId;
                                          }
                                    }

                                    if(GetDiskInfo(physicalDriveId, scsiPort,
scsiTargetId, interfaceType, vendorId))
                                    {
                                          int index = (int)vars.GetCount() - 1;
                                          if(! diskSize.IsEmpty())
                                          {
                                                DWORD totalDiskSize =
(DWORD)(_ttoi64(diskSize) / 1000 / 1000 - 50);
                                                if((totalDiskSize <
vars[index].TotalDiskSize - 100
                                                || vars[index].TotalDiskSize +
100 < totalDiskSize)
                                                && totalDiskSize >
vars[index].DiskSizeLba48
                                                )
                                                {
                                                      vars[index].TotalDiskSize =
totalDiskSize;
                                                }
                                          }

                                          BOOL flagSkipModelCheck = FALSE;

                                          vars[index].ModelWmi = model;
                                          // Модель
                                          model.Replace(_T(" SCSI Disk Device"),
_T(""));
                                          model.Replace(_T(" ATA Device"),
_T(""));

                                          if(model.Replace(_T(" USB Device"),
_T("")) > 0)
                                          {
                                                flagSkipModelCheck = TRUE;
                                                cstr.Format(_T("USB (%s)"),
vars[index].Interface);
                                                vars[index].Interface = cstr;
                                                vars[index].InterfaceType =
INTERFACE_TYPE_USB;

                                                for(int i = 0; i <
externals.GetCount(); i++)
                                                {
```

```
      if(externals.GetAt(i).Enclosure.Find(vars[index].ModelWmi) == 0)
                                                {
                                                      vars[index].Enclosure
= externals.GetAt(i).Enclosure;
                                                      vars[index].VendorId
= externals.GetAt(i).VendorId;
                                                      vars[index].ProductId
= externals.GetAt(i).ProductId;
                                                }
                                          }
                                    }
                                    else if(model.Replace(_T(" IEEE 1394
SBP2 Device"), _T("")) > 0)
                                    {
                                          flagSkipModelCheck = TRUE;
                                          cstr.Format(_T("IEEE 1394 (%s)"),
vars[index].Interface);
                                          vars[index].Interface = cstr;
                                          vars[index].InterfaceType =
INTERFACE_TYPE_IEEE1394;
                                          for(int i = 0; i <
externals.GetCount(); i++)
                                          {
      if(externals.GetAt(i).Enclosure.Find(vars[index].ModelWmi) == 0)
                                                {
                                                      vars[index].Enclosure
= externals.GetAt(i).Enclosure;
                                                      vars[index].VendorId
= externals.GetAt(i).VendorId;
                                                      vars[index].ProductId
= externals.GetAt(i).ProductId;
                                                }
                                          }
                                    }

                                    CString cmp, cmp1, cmp2, cmp3;
                                    cmp = model;
                                    cmp.Replace(_T(" "), _T(""));
                                    cmp1 = cmp.Left(16);

                                    cmp = vars[index].Model;
                                    cmp.Replace(_T(" "), _T(""));
                                    cmp2 = cmp.Left(16);

                                    cmp = vars[index].ModelReverse;
                                    cmp.Replace(_T(" "), _T(""));
                                    cmp3 = cmp.Left(16);

                                    if(vars[index].Model.IsEmpty())
                                    {
                                          vars.RemoveAt(index);
                                    }
                                    else if(flagSkipModelCheck)
                                    {
                                          // Не використовується
                                    }
                                    else if(model.IsEmpty() ||
cmp1.Compare(cmp2) == 0)
                                    {
                                          // Не використовується
                                    }
                                    else if(cmp1.Compare(cmp3) == 0)
                                    {
                                          vars[index].SerialNumber =
vars[index].SerialNumberReverse;
                                          vars[index].FirmwareRev =
vars[index].FirmwareRevReverse;
```

```
                                                vars[index].Model =
vars[index].ModelReverse;
                                                vars[index].ModelSerial =
vars[index].Model + vars[index].SerialNumber;

      vars[index].ModelSerial.Replace(_T("/"), _T(""));
                                          }
                                          else
                                          {
                                                vars.RemoveAt(index);
                                          }

                                          // РОЗШИФРУВАННЯ ЗНАЧЕНЬ
                                          // vars[index].VendorId =
VENDOR_MTRON;


                                                DebugPrint(_T("OK:Check Model Name"));
                                    }
                              }
                              catch(...)
                              {
                                    DebugPrint(_T("EX:GetDiskInfo pd=%d, sp=%d,
st=%d"));
                              }
                        }
                        SAFE_RELEASE(pEnumCOMDevs);
                        DebugPrint(_T("OK:SELECT * FROM Win32_DiskDrive"));
                  }
                  catch(...)
                  {
                        DebugPrint(_T("EX:SELECT * FROM Win32_DiskDrive"));
                  }

                  // Список дисків
                  try
                  {
                        DWORD driveLetterMap[256] = {0};

                        pIWbemServices->ExecQuery(SysAllocString(L"WQL"),
                              SysAllocString(L"SELECT * FROM
Win32_DiskPartition"), WBEM_FLAG_FORWARD_ONLY | WBEM_FLAG_RETURN_IMMEDIATELY,
NULL, &pEnumCOMDevs);
                        while(pEnumCOMDevs && SUCCEEDED(pEnumCOMDevs-
>Next(10000, 1, &pCOMDev, &uReturned)) && uReturned == 1)
                        {
                              DWORD physicalDriveId = 0;
                              CString partition, drive, mapping, cstr;
                              VARIANT pVal;
                              VariantClear(&pVal);
                              if(pCOMDev->Get(L"DeviceID", 0L, &pVal, NULL,
NULL) == WBEM_S_NO_ERROR && pVal.vt > VT_NULL)
                              {
                                    partition = pVal.bstrVal;
                                    VariantClear(&pVal);
                              }
                              cstr = partition;
                              cstr.Replace(_T("Disk #"), _T(""));
                              physicalDriveId = _ttoi(cstr);
                              SAFE_RELEASE(pCOMDev);

                              pIWbemServices->ExecQuery(SysAllocString(L"WQL"),
                                    SysAllocString(L"SELECT * FROM
Win32_LogicalDisk Where DriveType = 3"), WBEM_FLAG_FORWARD_ONLY |
WBEM_FLAG_RETURN_IMMEDIATELY, NULL, &pEnumCOMDevs2);
                              while(pEnumCOMDevs2 && SUCCEEDED(pEnumCOMDevs2-
>Next(10000, 1, &pCOMDev, &uReturned)) && uReturned == 1)
                              {
                                    VariantClear(&pVal);
```

```
                                    if(pCOMDev->Get(L"DeviceID", 0L, &pVal,
NULL, NULL) == WBEM_S_NO_ERROR && pVal.vt > VT_NULL)
                                    {
                                          drive = pVal.bstrVal;
                                          VariantClear(&pVal);
                                    }
                                    SAFE_RELEASE(pCOMDev);

                                    IWbemContext *pCtx = 0;
                                    IWbemCallResult *pResult = 0;

      mapping.Format(_T("Win32_LogicalDiskToPartition.Antecedent=\"Win32_DiskPar
tition.DeviceID=\\\"%s\\\"\",Dependent=\"Win32_LogicalDisk.DeviceID=\\\"%s\\\"\"
"),
                                          partition, drive);

                                    BSTR bstr;
                                    bstr = mapping.AllocSysString();
                                    pIWbemServices->GetObject(bstr, 0, pCtx,
&pCOMDev, &pResult);
                                    SysFreeString(bstr);
                                    if(pCOMDev)
                                    {
                                          driveLetterMap[physicalDriveId] |= 1
<< (drive.GetAt(0) - 'A');
                                    }
                                    SAFE_RELEASE(pCOMDev);
                              }
                              SAFE_RELEASE(pEnumCOMDevs2);
                        }
                        SAFE_RELEASE(pEnumCOMDevs);

                        for(int i = 0; i < vars.GetCount(); i++)
                        {
                              CString driveLetter = _T("");
                              for(int j = 0; j < 26; j++)
                              {
                                    if(driveLetterMap[vars[i].PhysicalDriveId] &
(1 << j))
                                    {
                                          CString cstr;
                                          cstr.Format(_T("%C"), j + 'A');
                                          driveLetter += cstr + _T(": ");
                                          vars[i].DriveLetterMap += (1 << j);
                                    }
                              }
                              vars[i].DriveMap.Append(driveLetter);
                        }
                        DebugPrint(_T("OK:Список дисків"));

                  }
                  catch(...)
                  {
                        DebugPrint(_T("EX:Список дисків"));
                  }

                  SAFE_RELEASE(pCOMDev);
                  SAFE_RELEASE(pEnumCOMDevs);
                  SAFE_RELEASE(pEnumCOMDevs2);
                  SAFE_RELEASE(pIWbemServices);
                  CoUninitialize();

                  DebugPrint(_T("OK:CoUninitialize()"));
            }
      }
      else
      {
            DebugPrint(_T("CAtaSmart::Init WMI off - Start"));
      }
```

```
	// \\\\.\\PhysicalDrive%d
	for(int i = 0; i < MAX_SEARCH_PHYSICAL_DRIVE; i++)
	{
		BOOL  bRet;
		HANDLE	hIoCtrl;
		DWORD dwReturned;
		DISK_GEOMETRY dg;

		hIoCtrl = GetIoCtrlHandle(i);
		if(hIoCtrl == INVALID_HANDLE_VALUE)
		{
			continue;
		}
		bRet = ::DeviceIoControl(hIoCtrl, IOCTL_DISK_GET_DRIVE_GEOMETRY,
			NULL, 0, &dg, sizeof(DISK_GEOMETRY),
			&dwReturned, NULL);
		::CloseHandle(hIoCtrl);
		if(bRet == FALSE || dwReturned != sizeof(DISK_GEOMETRY) ||
dg.MediaType != FixedMedia)
		{
			continue;
		}
// Визначення виробника
		if(GetDiskInfo(i, -1, -1, INTERFACE_TYPE_UNKNOWN, VENDOR_UNKNOWN))
		{
			int index = (int)vars.GetCount() - 1;
			CString cmp;
			cmp = vars[index].Model;
			if(cmp.Find(_T("DW C")) == 0 // WDC
			|| cmp.Find(_T("iHat")) == 0 // Hitachi
			|| cmp.Find(_T("ASSM")) == 0 // SAMSUNG
			|| cmp.Find(_T("aMtx")) == 0 // Maxtor
			|| cmp.Find(_T("OTHS")) == 0 // TOSHIBA
			|| cmp.Find(_T("UFIJ")) == 0 // FUJITSU
			)
			{
				vars[index].SerialNumber =
vars[index].SerialNumberReverse;
				vars[index].FirmwareRev =
vars[index].FirmwareRevReverse;
				vars[index].Model = vars[index].ModelReverse;
				vars[index].ModelSerial = vars[index].Model +
vars[index].SerialNumber;
				vars[index].ModelSerial.Replace(_T("/"), _T(""));
			}
		}
	}

	// сортування
	ATA_SMART_INFO* p = vars.GetData();
	qsort(p, vars.GetCount(), sizeof(ATA_SMART_INFO), Compare);

	DebugPrint(_T("OK:GetDiskInfo - PhysicalDrive"));

	// Визначення типу знайдених дисків
	if(advancedDiskSearch)
	{
		// \\\\.\\Scsi%d:
		for(int i = 0; i < MAX_SEARCH_SCSI_PORT; i++)
		{
			for(int j = 0; j < MAX_SEARCH_SCSI_TARGET_ID; j++)
			{
				if(GetDiskInfo(-1, i, j, INTERFACE_TYPE_UNKNOWN,
VENDOR_UNKNOWN))
				{
					int index = (int)vars.GetCount() - 1;
					CString cmp;
					cmp = vars[index].Model;
```

```
                                    if(cmp.Find(_T("DW C")) == 0 // WDC
                                    || cmp.Find(_T("iHat")) == 0 // Hitachi
                                    || cmp.Find(_T("ASSM")) == 0 // SAMSUNG
                                    || cmp.Find(_T("aMtx")) == 0 // Maxtor
                                    || cmp.Find(_T("OTHS")) == 0 // TOSHIBA
                                    || cmp.Find(_T("UFIJ")) == 0 // FUJITSU
                                    )
                                    {
                                          vars[index].SerialNumber =
vars[index].SerialNumberReverse;
                                          vars[index].FirmwareRev =
vars[index].FirmwareRevReverse;
                                          vars[index].Model =
vars[index].ModelReverse;
                                          vars[index].ModelSerial = vars[index].Model
+ vars[index].SerialNumber;
                                          vars[index].ModelSerial.Replace(_T("/"),
_T(""));
                                    }
                              }
                        }
                  }
                  DebugPrint(_T("OK:GetDiskInfo - Scsi"));
            }

            MeasuredGetTickCount = GetTickCount();
            DebugPrint(_T("CAtaSmart::Init - Complete"));

            if(flagChangeDisk != NULL)
            {
                  if(vars.GetCount() != previous.GetCount())
                  {
                        *flagChangeDisk = TRUE;
                  }
                  else
                  {
                        for(int i = 0; i < vars.GetCount(); i++)
                        {
                              if(vars.GetAt(i).PhysicalDriveId !=
previous.GetAt(i).PhysicalDriveId
                              || vars.GetAt(i).ScsiTargetId !=
previous.GetAt(i).ScsiTargetId
                              || vars.GetAt(i).ScsiPort != previous.GetAt(i).ScsiPort
                              )
                              {
                                    *flagChangeDisk = TRUE;
                                    break;
                              }
                        }
                  }
            }

            return IsEnabledWmi;
      }

      int CAtaSmart::Compare(const void *p1, const void *p2)
      {
            return ((ATA_SMART_INFO*)p1)->PhysicalDriveId - ((ATA_SMART_INFO*)p2)-
      >PhysicalDriveId;
      }

      BOOL CAtaSmart::AddDisk(INT physicalDriveId, INT scsiPort, INT scsiTargetId,
      COMMAND_TYPE commandType, IDENTIFY_DEVICE* identify)
      {
            ATA_SMART_INFO asi;

            memcpy(&(asi.IdentifyDevice), identify, sizeof(IDENTIFY_DEVICE));
            asi.PhysicalDriveId = physicalDriveId;
            asi.ScsiPort =  scsiPort;
```

```
asi.ScsiTargetId = scsiTargetId;
asi.CommandType = commandType;
asi.CommandTypeString = commandTypeString[commandType];

for(int i = 0; i < MAX_ATTRIBUTE; i++)
{
      ::ZeroMemory(&(asi.Attribute[i]), sizeof(SMART_ATTRIBUTE));
      ::ZeroMemory(&(asi.Threshold[i]), sizeof(SMART_THRESHOLD));
}

asi.IsSmartEnabled = FALSE;
asi.IsWord88 = FALSE;
asi.IsWord64_76 = FALSE;
asi.IsCheckSumError = FALSE;

asi.IsSmartSupported = FALSE;
asi.IsLba48Supported = FALSE;
asi.IsNcqSupported = FALSE;
asi.IsAamSupported = FALSE;
asi.IsApmSupported = FALSE;
asi.IsAamEnabled = FALSE;
asi.IsApmEnabled = FALSE;
asi.IsNvCacheSupported = FALSE;
asi.IsMaxtorMinute = FALSE;

asi.TotalDiskSize = 0;
asi.Cylinder = 0;
asi.Head = 0;
asi.Sector = 0;
asi.Sector28 = 0;
asi.Sector48 = 0;
asi.DiskSizeChs = 0;
asi.DiskSizeLba28 = 0;
asi.DiskSizeLba48 = 0;
asi.BufferSize = 0;
asi.NvCacheSize = 0;
asi.TransferModeType = 0;
asi.DetectedTimeUnitType = 0;
asi.MeasuredTimeUnitType = 0;
asi.AttributeCount = 0;
asi.DetectedPowerOnHours = -1;
asi.MeasuredPowerOnHours = -1;
asi.PowerOnRawValue = -1;
asi.PowerOnStartRawValue = -1;
asi.PowerOnCount = 0;
asi.Temperature = 0;
asi.Speed = 0.0;
asi.Life = -1;

asi.Major = 0;
asi.Minor = 0;

asi.DiskStatus = 0;
asi.DriveLetterMap = 0;

asi.AlarmTemperature = 0;

asi.VendorId = VENDOR_UNKNOWN;
asi.InterfaceType = INTERFACE_TYPE_UNKNOWN;

asi.VendorId = 0;
asi.ProductId = 0;

asi.SerialNumber = _T("");
asi.FirmwareRev = _T("");
asi.Model = _T("");
asi.ModelReverse = _T("");
asi.ModelWmi = _T("");
asi.ModelSerial = _T("");
```

```
	asi.DriveMap = _T("");
	asi.MaxTransferMode = _T("");
	asi.CurrentTransferMode = _T("");
	asi.MajorVersion = _T("");
	asi.MinorVersion = _T("");
	asi.Interface = _T("");
	asi.Enclosure = _T("");

	CHAR buf[64];

	// Перевірка помилок
	BYTE sum = 0;
	BYTE checkSum[IDENTIFY_BUFFER_SIZE];
	memcpy(checkSum, (void *)identify, IDENTIFY_BUFFER_SIZE);
	for(int j = 0; j < IDENTIFY_BUFFER_SIZE; j++)
	{
		sum += checkSum[j];
	}
	if(sum != 0)
	{
		asi.IsCheckSumError = TRUE;
	}

	// Обооротні дії
	strncpy_s(buf, 64, identify->SerialNumber, sizeof(identify-
>SerialNumber));
	asi.SerialNumberReverse = buf;
	asi.SerialNumberReverse.TrimLeft();
	asi.SerialNumberReverse.TrimRight();
	strncpy_s(buf, 64, identify->FirmwareRev, sizeof(identify->FirmwareRev));
	asi.FirmwareRevReverse = buf;
	asi.FirmwareRevReverse.TrimLeft();
	asi.FirmwareRevReverse.TrimRight();
	strncpy_s(buf, 64, identify->Model, sizeof(identify->Model));
	asi.ModelReverse = buf;
	asi.ModelReverse.TrimLeft();
	asi.ModelReverse.TrimRight();

	ChangeByteOrder(identify->SerialNumber, sizeof(identify->SerialNumber));
	ChangeByteOrder(identify->FirmwareRev, sizeof(identify->FirmwareRev));
	ChangeByteOrder(identify->Model, sizeof(identify->Model));

	if(CheckAsciiStringError(identify->SerialNumber, sizeof(identify-
>SerialNumber))
	|| CheckAsciiStringError(identify->FirmwareRev, sizeof(identify-
>FirmwareRev))
	|| CheckAsciiStringError(identify->Model, sizeof(identify->Model)))
	{
		return FALSE;
	}



	// Запис даних у структуру
	strncpy_s(buf, 64, identify->SerialNumber, sizeof(identify-
>SerialNumber));
	asi.SerialNumber = buf;
	asi.SerialNumber.TrimLeft();
	asi.SerialNumber.TrimRight();
	strncpy_s(buf, 64, identify->FirmwareRev, sizeof(identify->FirmwareRev));
	asi.FirmwareRev = buf;
	asi.FirmwareRev.TrimLeft();
	asi.FirmwareRev.TrimRight();
	strncpy_s(buf, 64, identify->Model, sizeof(identify->Model));
	asi.Model = buf;
	asi.Model.TrimLeft();
	asi.Model.TrimRight();

	if(asi.Model.IsEmpty() || asi.FirmwareRev.IsEmpty())
```

```
	{
		return FALSE;
	}

// РОЗШИФРУВАННЯ ЗНАЧЕНЬ
//    asi.Model = _T(" MTRON ") + asi.Model;

	asi.ModelSerial = asi.Model + asi.SerialNumber;
	asi.ModelSerial.Replace(_T("/"), _T(""));

	asi.Major = GetAtaMajorVersion(identify->MajorVersion, asi.MajorVersion);
	asi.TransferModeType = GetTransferMode(identify->MultiWordDma, identify-
>SerialAtaCapabilities,
                                       identify->UltraDmaMode,
asi.CurrentTransferMode, asi.MaxTransferMode,
                                       asi.Interface, &asi.InterfaceType);
	asi.DetectedTimeUnitType = GetTimeUnitType(asi.Model, asi.FirmwareRev,
asi.Major, asi.TransferModeType);

	// Feature
	if(asi.Major >= 3 && asi.IdentifyDevice.CommandSetSupported1 & (1 << 0))
	{
		asi.IsSmartSupported = TRUE;
	}
	if(asi.Major >= 3 && asi.IdentifyDevice.CommandSetSupported2 & (1 << 3))
	{
		asi.IsApmSupported = TRUE;
		if(asi.IdentifyDevice.CommandSetEnabled2 & (1 << 3))
		{
			asi.IsApmEnabled = TRUE;
		}
	}
	if(asi.Major >= 5 && asi.IdentifyDevice.CommandSetSupported2 & (1 << 9))
	{
		asi.IsAamSupported = TRUE;
		if(asi.IdentifyDevice.CommandSetEnabled2 & (1 << 9))
		{
			asi.IsAamEnabled = TRUE;
		}
	}

	if(asi.Major >= 5 && asi.IdentifyDevice.CommandSetSupported2 & (1 << 10))
	{
		asi.IsLba48Supported = TRUE;
	}

	if(asi.Major >= 6 && asi.IdentifyDevice.SerialAtaCapabilities & (1 << 8))
	{
		asi.IsNcqSupported = TRUE;
	}
	if(asi.Major >= 7 && asi.IdentifyDevice.NvCacheCapabilities & (1 << 0))
	{
		asi.IsNvCacheSupported = TRUE;
	}

	CString model = asi.Model;
	model.MakeUpper();
	if(model.Find(_T("MAXTOR")) == 0 && asi.DetectedTimeUnitType ==
POWER_ON_MINUTES)
	{
		asi.IsMaxtorMinute = TRUE;
	}

	// Розмір диску та розмір буферу
	asi.Cylinder = identify->LogicalCylinders;
	asi.Head = identify->LlogicalHeads;
	asi.Sector = identify->LogicalSectors;
	asi.Sector28 = identify->TotalAddressableSectors;
	asi.Sector48 = identify->MaxUserLba;
```

```
	asi.DiskSizeChs   = (DWORD)(((ULONGLONG)identify->LogicalCylinders *
identify->LlogicalHeads * identify->LogicalSectors * 512) / 1000 / 1000  - 50);
	asi.DiskSizeLba28 = (DWORD)(((ULONGLONG)identify->TotalAddressableSectors
* 512) / 1000 / 1000 - 50);
	if(asi.IsLba48Supported)
	{
		asi.DiskSizeLba48 = (DWORD)(((ULONGLONG)identify->MaxUserLba * 512)
/ 1000 / 1000 - 50);
	}
	asi.BufferSize = identify->BufferSize * 512;
	if(asi.IsNvCacheSupported)
	{
		asi.NvCacheSize = identify->NvCacheSizeLogicalBlocks * 512;
	}

	if(asi.DiskSizeChs == 0)
	{
		asi.TotalDiskSize = 0;
	}
	else if(asi.DiskSizeLba48 > asi.DiskSizeLba28)
	{
		asi.TotalDiskSize = asi.DiskSizeLba48;
	}
	else if(asi.DiskSizeLba28 > asi.DiskSizeChs)
	{
		asi.TotalDiskSize = asi.DiskSizeLba28;
	}
	else
	{
		asi.TotalDiskSize = asi.DiskSizeChs;
	}

	// Перевірка помилок для контролеру External ATA
	if(asi.IsLba48Supported && (identify->TotalAddressableSectors < 268435455
&& asi.DiskSizeLba28 != asi.DiskSizeLba48))
	{
		asi.DiskSizeLba48 = 0;
	}

	// SSD Життя
	if(asi.Model.Find(_T("MTRON")) == 0)
	{
		asi.VendorId = VENDOR_MTRON;
	}

	// перевірка підтримки S.M.A.R.T.
	switch(asi.CommandType)
	{
	case CMD_TYPE_PHYSICAL_DRIVE:
// РОЗШИФРУВАННЯ ЗНАЧЕНЬ
//		SendAtaCommandPd(physicalDriveId, 0xEF, 0x42, 0xFE);
//		SendAtaCommandPd(physicalDriveId, 0xEF, 0x05, 0xFE);

		if(GetSmartAttributePd(physicalDriveId, &asi))
		{
			GetSmartThresholdPd(physicalDriveId, &asi);
			asi.DiskStatus = CheckDiskStatus(asi.Attribute, asi.Threshold,
asi.AttributeCount, asi.VendorId);
			asi.IsSmartEnabled = TRUE;
		}
		else if(ControlSmartStatusPd(physicalDriveId, ENABLE_SMART))
		{
			if(GetSmartAttributePd(physicalDriveId, &asi))
			{
				GetSmartThresholdPd(physicalDriveId, &asi);
				asi.DiskStatus = CheckDiskStatus(asi.Attribute,
asi.Threshold, asi.AttributeCount, asi.VendorId);
				asi.IsSmartEnabled = TRUE;
			}
```

```
            }
            break;
      case CMD_TYPE_SCSI_MINIPORT:
// РОЗШИФРУВАННЯ ЗНАЧЕНЬ
//          SendAtaCommandScsi(scsiPort, scsiTargetId, 0xEC, 0x00, 0x00); // ID
//          SendAtaCommandScsi(scsiPort, scsiTargetId, 0xEF, 0x05, 0x80); // APM
//          SendAtaCommandScsi(scsiPort, scsiTargetId, 0xEF, 0x42, 0x80); // AAM

            if(GetSmartAttributeScsi(scsiPort, scsiTargetId, &asi))
            {
                  GetSmartThresholdScsi(scsiPort, scsiTargetId, &asi);
                  asi.DiskStatus = CheckDiskStatus(asi.Attribute, asi.Threshold,
asi.AttributeCount, asi.VendorId);
                  asi.IsSmartEnabled = TRUE;
            }
            else if(ControlSmartStatusScsi(scsiPort, scsiTargetId,
ENABLE_SMART))
            {
                  if(GetSmartAttributeScsi(scsiPort, scsiTargetId, &asi))
                  {
                        GetSmartThresholdScsi(scsiPort, scsiTargetId, &asi);
                        asi.DiskStatus = CheckDiskStatus(asi.Attribute,
asi.Threshold, asi.AttributeCount, asi.VendorId);
                        asi.IsSmartEnabled = TRUE;
                  }
            }
            break;
      case CMD_TYPE_SAT:
      case CMD_TYPE_SUNPLUS:
      case CMD_TYPE_IO_DATA:
      case CMD_TYPE_LOGITEC:
      case CMD_TYPE_JMICRON:
      case CMD_TYPE_CYPRESS:
// РОЗШИФРУВАННЯ ЗНАЧЕНЬ
//          SendAtaCommandSat(physicalDriveId, 0xEF, 0x05, 0xC0,
asi.CommandType);
//          SendAtaCommandSat(physicalDriveId, 0xEF, 0x42, 0xC0,
asi.CommandType);

            if(GetSmartAttributeSat(physicalDriveId, &asi))
            {
                  GetSmartThresholdSat(physicalDriveId, &asi);
                  asi.DiskStatus = CheckDiskStatus(asi.Attribute, asi.Threshold,
asi.AttributeCount, asi.VendorId);
                  asi.IsSmartEnabled = TRUE;
            }
            else if(ControlSmartStatusSat(physicalDriveId, ENABLE_SMART,
asi.CommandType))
            {
                  if(GetSmartAttributeSat(physicalDriveId, &asi))
                  {
                        GetSmartThresholdSat(physicalDriveId, &asi);
                        asi.DiskStatus = CheckDiskStatus(asi.Attribute,
asi.Threshold, asi.AttributeCount, asi.VendorId);
                        asi.IsSmartEnabled = TRUE;
                  }
            }
            break;
      default:
            return FALSE;
            break;
      }

      for(int i = 0; i < vars.GetCount(); i++)
      {
            if(asi.Model.Compare(vars[i].Model) == 0 &&
asi.SerialNumber.Compare(vars[i].SerialNumber) == 0)
            {
                  return FALSE;
```

```
			}
		}

		asi.PowerOnStartRawValue = asi.PowerOnRawValue;

		vars.Add(asi);

		return TRUE;
}

BOOL CAtaSmart::GetDiskInfo(INT physicalDriveId, INT scsiPort, INT scsiTargetId,
INTERFACE_TYPE interfaceType, VENDOR_ID vendorId)
{
		if(vars.GetCount() > MAX_DISK)
		{
			return FALSE;
		}
		// Перевірка перекриття
		for(int i = 0; i < vars.GetCount(); i++)
		{
			if(physicalDriveId >= 0 && vars[i].PhysicalDriveId ==
physicalDriveId)
			{
				return FALSE;
			}
			else if(scsiPort >= 0 && scsiTargetId >= 0
				&&	vars[i].ScsiPort == scsiPort && vars[i].ScsiTargetId ==
scsiTargetId)
			{
				return FALSE;
			}
		}

		IDENTIFY_DEVICE identify = {0};

		if(interfaceType == INTERFACE_TYPE_UNKNOWN || interfaceType ==
INTERFACE_TYPE_PATA || interfaceType == INTERFACE_TYPE_SATA)
		{
			if(physicalDriveId >= 0 && DoIdentifyDevicePd(physicalDriveId,
&identify))
			{
				return AddDisk(physicalDriveId, scsiPort, scsiTargetId,
CMD_TYPE_PHYSICAL_DRIVE, &identify);
			}
			else if(scsiPort >= 0 && scsiTargetId >= 0 &&
DoIdentifyDeviceScsi(scsiPort, scsiTargetId, &identify))
			{
				return AddDisk(physicalDriveId, scsiPort, scsiTargetId,
CMD_TYPE_SCSI_MINIPORT, &identify);
			}
		}
		else if(physicalDriveId >= 0)
		{
			/** РОЗШИФРУВАННЯ ЗНАЧЕНЬ
			if(TRUE)
			{
				DoIdentifyDeviceSat(physicalDriveId, &identify,
CMD_TYPE_DEBUG);
			}
			else
			*/
			if(interfaceType == INTERFACE_TYPE_USB && vendorId ==
USB_VENDOR_IO_DATA)
			{
				if(DoIdentifyDeviceSat(physicalDriveId, &identify,
CMD_TYPE_IO_DATA))
				{
					return AddDisk(physicalDriveId, scsiPort, scsiTargetId,
CMD_TYPE_IO_DATA, &identify);
```

```
                }
            }
            if(interfaceType == INTERFACE_TYPE_USB && vendorId ==
USB_VENDOR_LOGITEC)
            {
                if(DoIdentifyDeviceSat(physicalDriveId, &identify,
CMD_TYPE_LOGITEC))
                {
                    return AddDisk(physicalDriveId, scsiPort, scsiTargetId,
CMD_TYPE_LOGITEC, &identify);
                }
            }
            if(interfaceType == INTERFACE_TYPE_USB && vendorId ==
USB_VENDOR_SUNPLUS)
            {
                if(DoIdentifyDeviceSat(physicalDriveId, &identify,
CMD_TYPE_SUNPLUS))
                {
                    return AddDisk(physicalDriveId, scsiPort, scsiTargetId,
CMD_TYPE_SUNPLUS, &identify);
                }
            }
            else if(interfaceType == INTERFACE_TYPE_USB && vendorId ==
USB_VENDOR_CYPRESS)
            {
                if(DoIdentifyDeviceSat(physicalDriveId, &identify,
CMD_TYPE_CYPRESS))
                {
                    return AddDisk(physicalDriveId, scsiPort, scsiTargetId,
CMD_TYPE_CYPRESS, &identify);
                }
            }
            else if(interfaceType == INTERFACE_TYPE_USB &&
                (vendorId == USB_VENDOR_INITIO || vendorId ==
USB_VENDOR_OXFORD)
                )
            {
                if(DoIdentifyDeviceSat(physicalDriveId, &identify,
CMD_TYPE_SAT))
                {
                    return AddDisk(physicalDriveId, scsiPort, scsiTargetId,
CMD_TYPE_SAT, &identify);
                }
            }
            else
            {
                if(DoIdentifyDeviceSat(physicalDriveId, &identify,
CMD_TYPE_SAT))
                {
                    return AddDisk(physicalDriveId, scsiPort, scsiTargetId,
CMD_TYPE_SAT, &identify);
                }
                else if(DoIdentifyDeviceSat(physicalDriveId, &identify,
CMD_TYPE_JMICRON))
                {
                    return AddDisk(physicalDriveId, scsiPort, scsiTargetId,
CMD_TYPE_JMICRON, &identify);
                }
                else if(DoIdentifyDeviceSat(physicalDriveId, &identify,
CMD_TYPE_SUNPLUS))
                {
                    return AddDisk(physicalDriveId, scsiPort, scsiTargetId,
CMD_TYPE_SUNPLUS, &identify);
                }
                else if(DoIdentifyDeviceSat(physicalDriveId, &identify,
CMD_TYPE_CYPRESS))
                {
                    return AddDisk(physicalDriveId, scsiPort, scsiTargetId,
CMD_TYPE_CYPRESS, &identify);
```

```
                  }
                  else if(DoIdentifyDeviceSat(physicalDriveId, &identify,
CMD_TYPE_LOGITEC))
                  {
                        return AddDisk(physicalDriveId, scsiPort, scsiTargetId,
CMD_TYPE_LOGITEC, &identify);
                  }
            }
      }

      return FALSE;
}

/*-------------------------------------------------------------------------*/
// \\\\.\\Фізичний диск X
/*-------------------------------------------------------------------------*/
HANDLE CAtaSmart::GetIoCtrlHandle(BYTE index)
{
      CString     strDevice;
      strDevice.Format(_T("\\\\.\\PhysicalDrive%d"), index);

      return ::CreateFile(strDevice, GENERIC_READ | GENERIC_WRITE,
            FILE_SHARE_READ | FILE_SHARE_WRITE,
            NULL, OPEN_EXISTING, 0, NULL);
}

BOOL CAtaSmart::DoIdentifyDevicePd(INT physicalDriveId, IDENTIFY_DEVICE* data)
{
      BOOL  bRet;
      HANDLE      hIoCtrl;
      DWORD dwReturned;

      IDENTIFY_DEVICE_OUTDATA sendCmdOutParam;
      SENDCMDINPARAMS   sendCmd;

      if(data == NULL)
      {
            return      FALSE;
      }

      if(! SendAtaCommandPd(physicalDriveId, 0xEC, 0x00, 0x00, (PBYTE)data,
sizeof(IDENTIFY_DEVICE)))
      {
            ::ZeroMemory(data, sizeof(IDENTIFY_DEVICE));
            hIoCtrl = GetIoCtrlHandle(physicalDriveId);
            if(hIoCtrl == INVALID_HANDLE_VALUE)
            {
                  return      FALSE;
            }
            ::ZeroMemory(&sendCmdOutParam, sizeof(IDENTIFY_DEVICE_OUTDATA));
            ::ZeroMemory(&sendCmd, sizeof(SENDCMDINPARAMS));

            sendCmd.irDriveRegs.bCommandReg                 = ID_CMD;
            sendCmd.irDriveRegs.bSectorCountReg       = 1;
            sendCmd.irDriveRegs.bSectorNumberReg      = 1;
            sendCmd.cBufferSize                                   =
IDENTIFY_BUFFER_SIZE;

            bRet = ::DeviceIoControl(hIoCtrl, DFP_RECEIVE_DRIVE_DATA,
                  &sendCmd, sizeof(SENDCMDINPARAMS),
                  &sendCmdOutParam, sizeof(IDENTIFY_DEVICE_OUTDATA),
                  &dwReturned, NULL);

            ::CloseHandle(hIoCtrl);

            if(bRet == FALSE || dwReturned != sizeof(IDENTIFY_DEVICE_OUTDATA))
            {
                  return      FALSE;
            }
```

```
            memcpy_s(data, sizeof(IDENTIFY_DEVICE),
sendCmdOutParam.SendCmdOutParam.bBuffer, sizeof(IDENTIFY_DEVICE));
      }

      return      TRUE;
}

BOOL CAtaSmart::GetSmartAttributePd(INT PhysicalDriveId, ATA_SMART_INFO* asi)
{
      BOOL  bRet;
      HANDLE      hIoCtrl;
      DWORD dwReturned;

      SMART_READ_DATA_OUTDATA sendCmdOutParam;
      SENDCMDINPARAMS   sendCmd;

      hIoCtrl = GetIoCtrlHandle(PhysicalDriveId);
      if(hIoCtrl == INVALID_HANDLE_VALUE)
      {
            return      FALSE;
      }

      ::ZeroMemory(&sendCmdOutParam, sizeof(SMART_READ_DATA_OUTDATA));
      ::ZeroMemory(&sendCmd, sizeof(SENDCMDINPARAMS));

      sendCmd.irDriveRegs.bFeaturesReg    = READ_ATTRIBUTES;
      sendCmd.irDriveRegs.bSectorCountReg = 1;
      sendCmd.irDriveRegs.bSectorNumberReg= 1;
      sendCmd.irDriveRegs.bCylLowReg          = SMART_CYL_LOW;
      sendCmd.irDriveRegs.bCylHighReg         = SMART_CYL_HI;
      sendCmd.irDriveRegs.bCommandReg         = SMART_CMD;
      sendCmd.cBufferSize                           =
READ_ATTRIBUTE_BUFFER_SIZE;

      bRet = ::DeviceIoControl(hIoCtrl, DFP_RECEIVE_DRIVE_DATA,
            &sendCmd, sizeof(SENDCMDINPARAMS),
            &sendCmdOutParam, sizeof(SMART_READ_DATA_OUTDATA),
            &dwReturned, NULL);

      ::CloseHandle(hIoCtrl);

      if(bRet == FALSE || dwReturned != sizeof(SMART_READ_DATA_OUTDATA))
      {
            return      FALSE;
      }

      CString str;
      asi->AttributeCount = 0;
      int j = 0;
      for(int i = 0; i < MAX_ATTRIBUTE; i++)
      {
            DWORD rawValue = 0;
            memcpy(     &(asi->Attribute[j]),
                        &(sendCmdOutParam.Data[i * sizeof(SMART_ATTRIBUTE) +
1]), sizeof(SMART_ATTRIBUTE));

            if(asi->Attribute[j].Id != 0)
            {
                  switch(asi->Attribute[j].Id)
                  {
                  case 0x09: // Кількість відпрацьованих годин у включеному
стані
                        rawValue = MAKELONG(
                              MAKEWORD(asi->Attribute[j].RawValue[0], asi-
>Attribute[j].RawValue[1]),
                              MAKEWORD(asi->Attribute[j].RawValue[2], asi-
>Attribute[j].RawValue[3])
                              );
```

```
                        asi->PowerOnRawValue = rawValue;
                        asi->DetectedPowerOnHours = GetPowerOnHours(rawValue,
asi->DetectedTimeUnitType);
                        asi->MeasuredPowerOnHours = GetPowerOnHours(rawValue,
asi->MeasuredTimeUnitType);
                        break;
                  case 0x0C: // Кількість зафіксованих повторів
включення/вимикання живлення накопичувача
                        rawValue = MAKELONG(
                              MAKEWORD(asi->Attribute[j].RawValue[0], asi-
>Attribute[j].RawValue[1]),
                              MAKEWORD(asi->Attribute[j].RawValue[2], asi-
>Attribute[j].RawValue[3])
                              );
                        asi->PowerOnCount = rawValue;
                        break;
                  case 0xC2: // Температура
                        if(asi->Attribute[j].RawValue[0] > 0)
                        {
                              asi->Temperature = asi->Attribute[j].RawValue[0];
                        }
                        else
                        {
                              asi->Temperature = asi->Attribute[j].CurrentValue;
                        }
                  case 0xBB: // Визначення фірми-розробника
                        if(asi->VendorId == VENDOR_MTRON)
                        {
                              asi->Life = asi->Attribute[j].CurrentValue;
                        }
                        break;
                  default:
                        break;
                  }
                  j++;
            }
      }
      asi->AttributeCount = j;

      if(asi->AttributeCount > 0)
      {
            return TRUE;
      }
      else
      {
            return FALSE;
      }
}

BOOL CAtaSmart::GetSmartThresholdPd(INT physicalDriveId, ATA_SMART_INFO* asi)
{
      BOOL  bRet;
      HANDLE      hIoCtrl;
      DWORD dwReturned;

      SMART_READ_DATA_OUTDATA sendCmdOutParam;
      SENDCMDINPARAMS   sendCmd;

      hIoCtrl = GetIoCtrlHandle(physicalDriveId);
      if(hIoCtrl == INVALID_HANDLE_VALUE)
      {
            return      FALSE;
      }

      ::ZeroMemory(&sendCmdOutParam, sizeof(SMART_READ_DATA_OUTDATA));
      ::ZeroMemory(&sendCmd, sizeof(SENDCMDINPARAMS));

      sendCmd.irDriveRegs.bFeaturesReg    = READ_THRESHOLDS;
      sendCmd.irDriveRegs.bCylLowReg            = SMART_CYL_LOW;
```

```
	sendCmd.irDriveRegs.bCylHighReg		= SMART_CYL_HI;
	sendCmd.irDriveRegs.bCommandReg		= SMART_CMD;
	sendCmd.cBufferSize			=
READ_THRESHOLD_BUFFER_SIZE;

	bRet = ::DeviceIoControl(hIoCtrl, DFP_RECEIVE_DRIVE_DATA,
		&sendCmd, sizeof(SENDCMDINPARAMS),
		&sendCmdOutParam, sizeof(SMART_READ_DATA_OUTDATA),
		&dwReturned, NULL);

	::CloseHandle(hIoCtrl);

	if(bRet == FALSE || dwReturned != sizeof(SMART_READ_DATA_OUTDATA))
	{
		return	FALSE;
	}

	CString str;
	int j = 0;
	for(int i = 0; i < MAX_ATTRIBUTE; i++)
	{
		memcpy(	&(asi->Threshold[i]),
			&(sendCmdOutParam.Data[i * sizeof(SMART_THRESHOLD) +
1]), sizeof(SMART_THRESHOLD));

		if(asi->Threshold[j].Id != 0)
		{
			j++;
		}
	}

	return	TRUE;
}

BOOL CAtaSmart::ControlSmartStatusPd(INT physicalDriveId, BYTE command)
{
	BOOL  bRet;
	HANDLE	hIoCtrl;
	DWORD dwReturned;

	SENDCMDINPARAMS		sendCmd;
	SENDCMDOUTPARAMS  sendCmdOutParam;

	hIoCtrl = GetIoCtrlHandle(physicalDriveId);
	if(hIoCtrl == INVALID_HANDLE_VALUE)
	{
		return	FALSE;
	}

	::ZeroMemory(&sendCmd, sizeof(SENDCMDINPARAMS));
	::ZeroMemory(&sendCmdOutParam, sizeof(SENDCMDOUTPARAMS));

	sendCmd.irDriveRegs.bFeaturesReg    = command;
	sendCmd.irDriveRegs.bSectorCountReg = 1;
	sendCmd.irDriveRegs.bSectorNumberReg= 1;
	sendCmd.irDriveRegs.bCylLowReg		= SMART_CYL_LOW;
	sendCmd.irDriveRegs.bCylHighReg		= SMART_CYL_HI;
	sendCmd.irDriveRegs.bCommandReg		= SMART_CMD;
	sendCmd.cBufferSize			= 0;

	bRet = ::DeviceIoControl(hIoCtrl, DFP_SEND_DRIVE_COMMAND,
		&sendCmd, sizeof(SENDCMDINPARAMS) - 1,
		&sendCmdOutParam, sizeof(SENDCMDOUTPARAMS) -1,
		&dwReturned, NULL);

	::CloseHandle(hIoCtrl);

	return	bRet;
}
```

```
BOOL CAtaSmart::SendAtaCommandPd(INT physicalDriveId, BYTE main, BYTE sub, BYTE
param, PBYTE data, DWORD dataSize)
{
	BOOL  bRet;
	HANDLE	hIoCtrl;
	DWORD dwReturned;

	hIoCtrl = GetIoCtrlHandle(physicalDriveId);
	if(hIoCtrl == INVALID_HANDLE_VALUE)
	{
		return	FALSE;
	}

	if(m_FlagAtaPassThrough)
	{
		ATA_PASS_THROUGH_EX_WITH_BUFFERS ab;
		::ZeroMemory(&ab, sizeof(ab));
		ab.Apt.Length = sizeof(ATA_PASS_THROUGH_EX);
		ab.Apt.TimeOutValue = 10;
		DWORD size = offsetof(ATA_PASS_THROUGH_EX_WITH_BUFFERS, Buf);
		ab.Apt.DataBufferOffset = size;

		if(dataSize > 0)
		{
			if(dataSize > sizeof(ab.Buf))
			{
				return FALSE;
			}
			ab.Apt.AtaFlags = ATA_FLAGS_DATA_IN;
			ab.Apt.DataTransferLength = dataSize;
			size += dataSize;
		}

		ab.Apt.CurrentTaskFile.bFeaturesReg = sub;
		ab.Apt.CurrentTaskFile.bSectorCountReg = param;
		ab.Apt.CurrentTaskFile.bCommandReg = main;

		bRet = ::DeviceIoControl(hIoCtrl, IOCTL_ATA_PASS_THROUGH,
			&ab, size, &ab, size, &dwReturned, NULL);
		::CloseHandle(hIoCtrl);
		if(bRet && dataSize && data != NULL)
		{
			memcpy_s(data, dataSize, ab.Buf, dataSize);
		}
	}
	else if(m_Os.dwMajorVersion == 4)
	{
		return FALSE;
	}
	else
	{
		DWORD size = sizeof(CMD_IDE_PATH_THROUGH) - 1 + dataSize;
		CMD_IDE_PATH_THROUGH* buf =
  (CMD_IDE_PATH_THROUGH*)VirtualAlloc(NULL, size, MEM_COMMIT, PAGE_READWRITE);

		buf->reg.bFeaturesReg		= sub;
		buf->reg.bSectorCountReg	= param;
		buf->reg.bSectorNumberReg	= 0;
		buf->reg.bCylLowReg			= 0;
		buf->reg.bCylHighReg		= 0;
		buf->reg.bDriveHeadReg		= 0;
		buf->reg.bCommandReg		= main;
		buf->reg.bReserved			= 0;
		buf->length					= dataSize;

		bRet = ::DeviceIoControl(hIoCtrl, IOCTL_IDE_PASS_THROUGH,
			buf, size, buf, size, &dwReturned, NULL);
		::CloseHandle(hIoCtrl);
```

```
            if(bRet && dataSize && data != NULL)
            {
                  memcpy_s(data, dataSize, buf->buffer, dataSize);
            }
            VirtualFree(buf, 0, MEM_RELEASE);
      }

      return      bRet;
}

/*-------------------------------------------------------------------------*/
//  \\\\.\\ Диск SCSI X
/*-------------------------------------------------------------------------*/

BOOL CAtaSmart::DoIdentifyDeviceScsi(INT scsiPort, INT scsiTargetId,
IDENTIFY_DEVICE* identify)
{
      int done = FALSE;
      int controller = 0;
      int current = 0;
      HANDLE hScsiDriveIOCTL = 0;
      CString driveName;
      driveName.Format(_T("\\\\.\\Scsi%d:"), scsiPort);
      hScsiDriveIOCTL = CreateFile(driveName, GENERIC_READ | GENERIC_WRITE,
                                          FILE_SHARE_READ | FILE_SHARE_WRITE,
NULL, OPEN_EXISTING, 0, NULL);
      if(hScsiDriveIOCTL != INVALID_HANDLE_VALUE)
      {
            BYTE buffer[sizeof(SRB_IO_CONTROL) + sizeof(SENDCMDOUTPARAMS) +
IDENTIFY_BUFFER_SIZE];
            SRB_IO_CONTROL *p = (SRB_IO_CONTROL *)buffer;
            SENDCMDINPARAMS *pin = (SENDCMDINPARAMS *)(buffer +
sizeof(SRB_IO_CONTROL));
            DWORD dummy;

            memset(buffer, 0, sizeof(buffer));
            p->HeaderLength = sizeof (SRB_IO_CONTROL);
            p->Timeout = 2;
            p->Length = sizeof(SENDCMDOUTPARAMS) + IDENTIFY_BUFFER_SIZE;
            p->ControlCode = IOCTL_SCSI_MINIPORT_IDENTIFY;
            memcpy((char *)p->Signature, "SCSIDISK", 8);
            pin->irDriveRegs.bCommandReg = ID_CMD;
            pin->bDriveNumber = scsiTargetId;

            if(DeviceIoControl(hScsiDriveIOCTL, IOCTL_SCSI_MINIPORT,
                                          buffer, sizeof(SRB_IO_CONTROL) +
sizeof(SENDCMDINPARAMS) - 1,
                                          buffer, sizeof(SRB_IO_CONTROL) +
sizeof(SENDCMDOUTPARAMS) + IDENTIFY_BUFFER_SIZE,
                                          &dummy, NULL))
            {
                  SENDCMDOUTPARAMS *pOut = (SENDCMDOUTPARAMS *)(buffer +
sizeof(SRB_IO_CONTROL));
                  if(*(pOut->bBuffer) > 0)
                  {
                        done = TRUE;
                        memcpy_s(identify, sizeof(IDENTIFY_DEVICE), pOut-
>bBuffer, sizeof(IDENTIFY_DEVICE));
                  }
            }
            CloseHandle(hScsiDriveIOCTL);
      }
      return done;
}

BOOL CAtaSmart::GetSmartAttributeScsi(INT scsiPort, INT scsiTargetId,
ATA_SMART_INFO* asi)
{
      HANDLE hScsiDriveIOCTL = 0;
```

```
	CString driveName;
	driveName.Format(_T("\\\\.\\Scsi%d:"), scsiPort);
	hScsiDriveIOCTL = CreateFile(driveName, GENERIC_READ | GENERIC_WRITE,
                                     FILE_SHARE_READ | FILE_SHARE_WRITE,
NULL, OPEN_EXISTING, 0, NULL);
	if(hScsiDriveIOCTL != INVALID_HANDLE_VALUE)
	{
		BYTE buffer[sizeof(SRB_IO_CONTROL) + sizeof(SENDCMDOUTPARAMS) +
READ_ATTRIBUTE_BUFFER_SIZE];
		SRB_IO_CONTROL *p = (SRB_IO_CONTROL *)buffer;
		SENDCMDINPARAMS *pin = (SENDCMDINPARAMS *)(buffer +
sizeof(SRB_IO_CONTROL));
		DWORD dummy;
		memset(buffer, 0, sizeof(buffer));
		p->HeaderLength = sizeof(SRB_IO_CONTROL);
		p->Timeout = 2;
		p->Length = sizeof(SENDCMDOUTPARAMS) + READ_ATTRIBUTE_BUFFER_SIZE;
		p->ControlCode = IOCTL_SCSI_MINIPORT_READ_SMART_ATTRIBS;
		memcpy((char *)p->Signature, "SCSIDISK", 8);
		pin->irDriveRegs.bFeaturesReg      = READ_ATTRIBUTES;
		pin->irDriveRegs.bSectorCountReg   = 1;
		pin->irDriveRegs.bSectorNumberReg  = 1;
		pin->irDriveRegs.bCylLowReg              = SMART_CYL_LOW;
		pin->irDriveRegs.bCylHighReg       = SMART_CYL_HI;
		pin->irDriveRegs.bCommandReg       = SMART_CMD;
		pin->cBufferSize                         =
READ_ATTRIBUTE_BUFFER_SIZE;
		pin->bDriveNumber                        = scsiTargetId;

		if(DeviceIoControl(hScsiDriveIOCTL, IOCTL_SCSI_MINIPORT,
                                     buffer, sizeof(SRB_IO_CONTROL) +
sizeof(SENDCMDINPARAMS) - 1,
                                     buffer, sizeof(SRB_IO_CONTROL) +
sizeof(SENDCMDOUTPARAMS) + READ_ATTRIBUTE_BUFFER_SIZE,
                                     &dummy, NULL))
		{
			SENDCMDOUTPARAMS *pOut = (SENDCMDOUTPARAMS *)(buffer +
sizeof(SRB_IO_CONTROL));
			if(*(pOut->bBuffer) > 0)
			{
				CString str;
				asi->AttributeCount = 0;
				int j = 0;

				for(int i = 0; i < MAX_ATTRIBUTE; i++)
				{
					DWORD rawValue = 0;

					memcpy(     &(asi->Attribute[j]),
                                                &(pOut->bBuffer[i * sizeof(SMART_ATTRIBUTE)
+ 2]), sizeof(SMART_ATTRIBUTE));

					if(asi->Attribute[j].Id != 0)
					{
                                                switch(asi->Attribute[j].Id)
                                                {
                                                case 0x09: // Кількість відпрацьованих годин
у включеному стані
                                                        rawValue = MAKELONG(
                                                                MAKEWORD(asi-
>Attribute[j].RawValue[0], asi->Attribute[j].RawValue[1]),
                                                                MAKEWORD(asi-
>Attribute[j].RawValue[2], asi->Attribute[j].RawValue[3])
                                                                );
                                                        asi->PowerOnRawValue = rawValue;
                                                        asi->DetectedPowerOnHours =
GetPowerOnHours(rawValue, asi->DetectedTimeUnitType);
                                                        asi->MeasuredPowerOnHours =
GetPowerOnHours(rawValue, asi->MeasuredTimeUnitType);
```

```
                                    break;
                                case 0x0C: // Кількість зафіксованих
повторів включення/вимикання живлення накопичувача
                                    rawValue = MAKELONG(
                                          MAKEWORD(asi-
>Attribute[j].RawValue[0], asi->Attribute[j].RawValue[1]),
                                          MAKEWORD(asi-
>Attribute[j].RawValue[2], asi->Attribute[j].RawValue[3])
                                          );
                                    asi->PowerOnCount = rawValue;
                                    break;
                                case 0xC2: // Температура
                                    if(asi->Attribute[j].RawValue[0] > 0)
                                    {
                                          asi->Temperature = asi-
>Attribute[j].RawValue[0];
                                    }
                                    else
                                    {
                                          asi->Temperature = asi-
>Attribute[j].CurrentValue;
                                    }
                                    break;
                                case 0xBB: // Визначення фірми-розробника
                                    if(asi->VendorId == VENDOR_MTRON)
                                    {
                                          asi->Life = asi-
>Attribute[j].CurrentValue;
                                    }
                                    break;
                                default:
                                    break;
                                }
                                j++;
                          }
                    }
                    asi->AttributeCount = j;
              }
        }
    }
    CloseHandle(hScsiDriveIOCTL);

    if(asi->AttributeCount > 0)
    {
        return     TRUE;
    }
    else
    {
        return FALSE;
    }
}

BOOL CAtaSmart::GetSmartThresholdScsi(INT scsiPort, INT scsiTargetId,
ATA_SMART_INFO* asi)
{
    HANDLE hScsiDriveIOCTL = 0;
    CString driveName;
    driveName.Format(_T("\\\\.\\Scsi%d:"), scsiPort);
    hScsiDriveIOCTL = CreateFile(driveName, GENERIC_READ | GENERIC_WRITE,
                                    FILE_SHARE_READ | FILE_SHARE_WRITE,
NULL, OPEN_EXISTING, 0, NULL);
    if(hScsiDriveIOCTL != INVALID_HANDLE_VALUE)
    {
        BYTE buffer[sizeof(SRB_IO_CONTROL) + sizeof(SENDCMDOUTPARAMS) +
READ_THRESHOLD_BUFFER_SIZE];
        SRB_IO_CONTROL *p = (SRB_IO_CONTROL *)buffer;
        SENDCMDINPARAMS *pin = (SENDCMDINPARAMS *)(buffer +
sizeof(SRB_IO_CONTROL));
        DWORD dummy;
```

```
            memset(buffer, 0, sizeof(buffer));
            p->HeaderLength = sizeof(SRB_IO_CONTROL);
            p->Timeout = 2;
            p->Length = sizeof(SENDCMDOUTPARAMS) + READ_THRESHOLD_BUFFER_SIZE;
            p->ControlCode = IOCTL_SCSI_MINIPORT_READ_SMART_THRESHOLDS;
            memcpy((char *)p->Signature, "SCSIDISK", 8);
            pin->irDriveRegs.bFeaturesReg      = READ_THRESHOLDS;
            pin->irDriveRegs.bSectorCountReg   = 1;
            pin->irDriveRegs.bSectorNumberReg  = 1;
            pin->irDriveRegs.bCylLowReg              = SMART_CYL_LOW;
            pin->irDriveRegs.bCylHighReg       = SMART_CYL_HI;
            pin->irDriveRegs.bCommandReg       = SMART_CMD;
            pin->cBufferSize                         =
READ_THRESHOLD_BUFFER_SIZE;
            pin->bDriveNumber                        = scsiTargetId;

            if(DeviceIoControl(hScsiDriveIOCTL, IOCTL_SCSI_MINIPORT,
                                                buffer, sizeof(SRB_IO_CONTROL) +
sizeof(SENDCMDINPARAMS) - 1,
                                                buffer, sizeof(SRB_IO_CONTROL) +
sizeof(SENDCMDOUTPARAMS) + READ_THRESHOLD_BUFFER_SIZE,
                                                &dummy, NULL))
            {
                  SENDCMDOUTPARAMS *pOut = (SENDCMDOUTPARAMS *)(buffer +
sizeof(SRB_IO_CONTROL));
                  if(*(pOut->bBuffer) > 0)
                  {
                        int j = 0;
                        for(int i = 0; i < MAX_ATTRIBUTE; i++)
                        {
                              memcpy(    &(asi->Threshold[j]),
                                    &(pOut->bBuffer[i * sizeof(SMART_THRESHOLD)
+ 2]), sizeof(SMART_THRESHOLD));
                              if(asi->Threshold[j].Id != 0)
                              {
                                    j++;
                              }
                        }
                  }
            }
      }
      CloseHandle (hScsiDriveIOCTL);

      if(asi->AttributeCount > 0)
      {
            return TRUE;
      }
      else
      {
            return FALSE;
      }
}

BOOL CAtaSmart::ControlSmartStatusScsi(INT scsiPort, INT scsiTargetId, BYTE
command)
{
      BOOL  bRet;
      HANDLE hScsiDriveIOCTL = 0;
      CString driveName;
      driveName.Format(_T("\\\\.\\Scsi%d:"), scsiPort);
      hScsiDriveIOCTL = CreateFile(driveName, GENERIC_READ | GENERIC_WRITE,
                                          FILE_SHARE_READ | FILE_SHARE_WRITE,
NULL, OPEN_EXISTING, 0, NULL);
      if(hScsiDriveIOCTL != INVALID_HANDLE_VALUE)
      {
            BYTE buffer[sizeof(SRB_IO_CONTROL) + sizeof(SENDCMDOUTPARAMS) +
SCSI_MINIPORT_BUFFER_SIZE];
            SRB_IO_CONTROL *p = (SRB_IO_CONTROL *)buffer;
```

```
            SENDCMDINPARAMS *pin = (SENDCMDINPARAMS *)(buffer +
sizeof(SRB_IO_CONTROL));
            DWORD dummy;
            memset(buffer, 0, sizeof(buffer));
            p->HeaderLength = sizeof(SRB_IO_CONTROL);
            p->Timeout = 2;
            p->Length = sizeof(SENDCMDOUTPARAMS) + SCSI_MINIPORT_BUFFER_SIZE;
            if(command == DISABLE_SMART)
            {
                  p->ControlCode = IOCTL_SCSI_MINIPORT_DISABLE_SMART;
            }
            else
            {
                  p->ControlCode = IOCTL_SCSI_MINIPORT_ENABLE_SMART;
            }
            memcpy((char *)p->Signature, "SCSIDISK", 8);
            pin->irDriveRegs.bFeaturesReg       = command;
            pin->irDriveRegs.bSectorCountReg    = 1;
            pin->irDriveRegs.bSectorNumberReg   = 1;
            pin->irDriveRegs.bCylLowReg               = SMART_CYL_LOW;
            pin->irDriveRegs.bCylHighReg        = SMART_CYL_HI;
            pin->irDriveRegs.bCommandReg        = SMART_CMD;
            pin->cBufferSize                          =
SCSI_MINIPORT_BUFFER_SIZE;
            pin->bDriveNumber                         = scsiTargetId;

            bRet = DeviceIoControl(hScsiDriveIOCTL, IOCTL_SCSI_MINIPORT,
                                                buffer, sizeof(SRB_IO_CONTROL) +
sizeof(SENDCMDINPARAMS) - 1,
                                                buffer, sizeof(SRB_IO_CONTROL) +
sizeof(SENDCMDOUTPARAMS) + SCSI_MINIPORT_BUFFER_SIZE,
                                                &dummy, NULL);

      }
      CloseHandle(hScsiDriveIOCTL);

      return bRet;
}


BOOL CAtaSmart::SendAtaCommandScsi(INT scsiPort, INT scsiTargetId, BYTE main,
BYTE sub, BYTE param)
{
/** Does not work...
      BOOL  bRet;
      HANDLE hScsiDriveIOCTL = 0;
      CString driveName;
      driveName.Format(_T("\\\\.\\Scsi%d:"), scsiPort);
      hScsiDriveIOCTL = CreateFile(driveName, GENERIC_READ | GENERIC_WRITE,
                                          FILE_SHARE_READ | FILE_SHARE_WRITE,
NULL, OPEN_EXISTING, 0, NULL);
      if(hScsiDriveIOCTL != INVALID_HANDLE_VALUE)
      {
            DWORD dummy;
            CMD_ATA_PASS_THROUGH_WITH_BUFFERS capt;
            ::ZeroMemory(&capt, sizeof(CMD_ATA_PASS_THROUGH_WITH_BUFFERS));
            capt.apt.Length = sizeof(CMD_ATA_PASS_THROUGH);
            capt.apt.PathId = 0;
            capt.apt.TargetId = 0;
            capt.apt.Lun = 0;
            capt.apt.TimeOutValue = 10;

            DWORD size = offsetof(CMD_ATA_PASS_THROUGH_WITH_BUFFERS, DataBuf);
            capt.apt.DataBufferOffset = size;

            capt.apt.AtaFlags = 0x02;
            capt.apt.DataTransferLength = 512;
            size += 512;
            capt.DataBuf[0] = 0xCF;
```

```
            capt.apt.CurrentTaskFile.bFeaturesReg= sub;
            capt.apt.CurrentTaskFile.bSectorCountReg = param;
            capt.apt.CurrentTaskFile.bDriveHeadReg = 0xA0;
            capt.apt.CurrentTaskFile.bCommandReg = main;

            bRet = DeviceIoControl(hScsiDriveIOCTL, IOCTL_ATA_PASS_THROUGH,
                                                &capt, size,
                                                &capt, size,
                                                &dummy, NULL);
      }
      CloseHandle(hScsiDriveIOCTL);

      return bRet;
*/
      return FALSE;
}

/*-------------------------------------------------------------------------*/
//  SCSI / ATA переклад (SAT)
/*-------------------------------------------------------------------------*/

BOOL CAtaSmart::DoIdentifyDeviceSat(INT physicalDriveId, IDENTIFY_DEVICE* data,
COMMAND_TYPE type)
{
      BOOL  bRet;
      HANDLE      hIoCtrl;
      DWORD dwReturned;
      DWORD length;

      SCSI_PASS_THROUGH_WITH_BUFFERS sptwb;

      if(data == NULL)
      {
            return      FALSE;
      }

      ::ZeroMemory(data, sizeof(IDENTIFY_DEVICE));

      hIoCtrl = GetIoCtrlHandle(physicalDriveId);
      if(hIoCtrl == INVALID_HANDLE_VALUE)
      {
            return      FALSE;
      }

      ::ZeroMemory(&sptwb,sizeof(SCSI_PASS_THROUGH_WITH_BUFFERS));

    sptwb.Spt.Length = sizeof(SCSI_PASS_THROUGH);
    sptwb.Spt.PathId = 0;
    sptwb.Spt.TargetId = 0;
    sptwb.Spt.Lun = 0;
    sptwb.Spt.SenseInfoLength = 24;
    sptwb.Spt.DataIn = SCSI_IOCTL_DATA_IN;
    sptwb.Spt.DataTransferLength = IDENTIFY_BUFFER_SIZE;
    sptwb.Spt.TimeOutValue = 2;
    sptwb.Spt.DataBufferOffset = offsetof(SCSI_PASS_THROUGH_WITH_BUFFERS,
DataBuf);
    sptwb.Spt.SenseInfoOffset = offsetof(SCSI_PASS_THROUGH_WITH_BUFFERS,
SenseBuf);

      if(type == CMD_TYPE_SAT)
      {
            sptwb.Spt.CdbLength = 12;
            sptwb.Spt.Cdb[0] = 0xA1;//ATA проходів через(12) операцій коду(A1h)
            sptwb.Spt.Cdb[1] = (4 << 1) | 0; //MULTIPLE_COUNT=0,PROTOCOL=4(PIO
Data-In),Reserved
            sptwb.Spt.Cdb[2] = (1 << 3) | (1 << 2) |
2;//OFF_LINE=0,CK_COND=0,Reserved=0,T_DIR=1(ToDevice),BYTE_BLOCK=1,T_LENGTH=2
            sptwb.Spt.Cdb[3] = 0;//FEATURES (7:0)
            sptwb.Spt.Cdb[4] = 1;//SECTOR_COUNT (7:0)
```

```
        sptwb.Spt.Cdb[5] = 0;//LBA_LOW (7:0)
        sptwb.Spt.Cdb[6] = 0;//LBA_MID (7:0)
        sptwb.Spt.Cdb[7] = 0;//LBA_HIGH (7:0)
        sptwb.Spt.Cdb[9] = ID_CMD;//COMMAND
    }
    else if(type == CMD_TYPE_SUNPLUS)
    {
        sptwb.Spt.CdbLength = 12;
        sptwb.Spt.Cdb[0] = 0xF8;
        sptwb.Spt.Cdb[1] = 0x00;
        sptwb.Spt.Cdb[2] = 0x22;
        sptwb.Spt.Cdb[3] = 0x10;
        sptwb.Spt.Cdb[4] = 0x01;
        sptwb.Spt.Cdb[5] = 0x00;
        sptwb.Spt.Cdb[6] = 0x01;
        sptwb.Spt.Cdb[7] = 0x00;
        sptwb.Spt.Cdb[8] = 0x00;
        sptwb.Spt.Cdb[9] = 0x00;
        sptwb.Spt.Cdb[10] = 0xA0;
        sptwb.Spt.Cdb[11] = 0xEC; // ID_CMD
    }
    else if(type == CMD_TYPE_IO_DATA)
    {
        sptwb.Spt.CdbLength = 12;
        sptwb.Spt.Cdb[0] = 0xE3;
        sptwb.Spt.Cdb[1] = 0x00;
        sptwb.Spt.Cdb[2] = 0x00;
        sptwb.Spt.Cdb[3] = 0x01;
        sptwb.Spt.Cdb[4] = 0x01;
        sptwb.Spt.Cdb[5] = 0x00;
        sptwb.Spt.Cdb[6] = 0x00;
        sptwb.Spt.Cdb[7] = 0xA0;
        sptwb.Spt.Cdb[8] = 0xEC;  // ID_CMD
        sptwb.Spt.Cdb[9] = 0x00;
        sptwb.Spt.Cdb[10] = 0x00;
        sptwb.Spt.Cdb[11] = 0x00;
    }
    else if(type == CMD_TYPE_LOGITEC)
    {
        sptwb.Spt.CdbLength = 10;
        sptwb.Spt.Cdb[0] = 0xE0;
        sptwb.Spt.Cdb[1] = 0x00;
        sptwb.Spt.Cdb[2] = 0x00;
        sptwb.Spt.Cdb[3] = 0x00;
        sptwb.Spt.Cdb[4] = 0x00;
        sptwb.Spt.Cdb[5] = 0x00;
        sptwb.Spt.Cdb[6] = 0x00;
        sptwb.Spt.Cdb[7] = 0xA0;
        sptwb.Spt.Cdb[8] = 0xEC;  // ID_CMD
        sptwb.Spt.Cdb[9] = 0x4C;
    }
    else if(type == CMD_TYPE_JMICRON)
    {
        sptwb.Spt.CdbLength = 12;
        sptwb.Spt.Cdb[0] = 0xDF;
        sptwb.Spt.Cdb[1] = 0x10;
        sptwb.Spt.Cdb[2] = 0x00;
        sptwb.Spt.Cdb[3] = 0x02;
        sptwb.Spt.Cdb[4] = 0x00;
        sptwb.Spt.Cdb[5] = 0x00;
        sptwb.Spt.Cdb[6] = 0x01;
        sptwb.Spt.Cdb[7] = 0x00;
        sptwb.Spt.Cdb[8] = 0x00;
        sptwb.Spt.Cdb[9] = 0x00;
        sptwb.Spt.Cdb[10] = 0xA0;
        sptwb.Spt.Cdb[11] = 0xEC; // ID_CMD
    }
    else if(type == CMD_TYPE_CYPRESS)
    {
```

```
		sptwb.Spt.CdbLength = 16;
		sptwb.Spt.Cdb[0] = 0x24;
		sptwb.Spt.Cdb[1] = 0x24;
		sptwb.Spt.Cdb[2] = 0x00;
		sptwb.Spt.Cdb[3] = 0xBE;
		sptwb.Spt.Cdb[4] = 0x01;
		sptwb.Spt.Cdb[5] = 0x00;
		sptwb.Spt.Cdb[6] = 0x00;
		sptwb.Spt.Cdb[7] = 0x01;
		sptwb.Spt.Cdb[8] = 0x00;
		sptwb.Spt.Cdb[9] = 0x00;
		sptwb.Spt.Cdb[10] = 0x00;
		sptwb.Spt.Cdb[11] = 0xA0;
		sptwb.Spt.Cdb[12] = 0xEC; // ID_CMD
		sptwb.Spt.Cdb[13] = 0x00;
		sptwb.Spt.Cdb[14] = 0x00;
		sptwb.Spt.Cdb[15] = 0x00;
	}
/*
	else if(type == CMD_TYPE_DEBUG)
	{
		sptwb.Spt.CdbLength = 16;
		for(int i = 0xA0; i <= 0xFF; i++)
		{
			for(int j = 8; j < 16; j++)
			{
				::ZeroMemory(&sptwb.Spt.Cdb, 16);
				sptwb.Spt.Cdb[0] = i;
				sptwb.Spt.Cdb[j - 1] = 0xA0;
				sptwb.Spt.Cdb[j] = 0xEC;

				length = offsetof(SCSI_PASS_THROUGH_WITH_BUFFERS,
DataBuf) + sptwb.Spt.DataTransferLength;

				bRet = ::DeviceIoControl(hIoCtrl,
IOCTL_SCSI_PASS_THROUGH,
					&sptwb, sizeof(SCSI_PASS_THROUGH),
					&sptwb, length,   &dwReturned, NULL);


				if(bRet == FALSE || dwReturned != length)
				{
					continue;
				}

				CString cstr;
				cstr.Format(_T("i = %d, j = %d"), i, j);
				AfxMessageBox(cstr);

				::CloseHandle(hIoCtrl);
				memcpy_s(data, sizeof(IDENTIFY_DEVICE), sptwb.DataBuf,
sizeof(IDENTIFY_DEVICE));

				return TRUE;
			}
		}
	}
*/
	else
	{
		return FALSE;
	}

	length = offsetof(SCSI_PASS_THROUGH_WITH_BUFFERS, DataBuf) +
sptwb.Spt.DataTransferLength;

	bRet = ::DeviceIoControl(hIoCtrl, IOCTL_SCSI_PASS_THROUGH,
		&sptwb, sizeof(SCSI_PASS_THROUGH),
		&sptwb, length,   &dwReturned, NULL);
```

```
	::CloseHandle(hIoCtrl);

	if(bRet == FALSE || dwReturned != length)
	{
		return	FALSE;
	}

	memcpy_s(data, sizeof(IDENTIFY_DEVICE), sptwb.DataBuf,
sizeof(IDENTIFY_DEVICE));

	return	TRUE;
}

BOOL CAtaSmart::GetSmartAttributeSat(INT PhysicalDriveId, ATA_SMART_INFO* asi)
{
	BOOL  bRet;
	HANDLE	hIoCtrl;
	DWORD dwReturned;
	DWORD length;

	SCSI_PASS_THROUGH_WITH_BUFFERS sptwb;

	hIoCtrl = GetIoCtrlHandle(PhysicalDriveId);
	if(hIoCtrl == INVALID_HANDLE_VALUE)
	{
		return	FALSE;
	}

	::ZeroMemory(&sptwb,sizeof(SCSI_PASS_THROUGH_WITH_BUFFERS));

    sptwb.Spt.Length = sizeof(SCSI_PASS_THROUGH);
    sptwb.Spt.PathId = 0;
    sptwb.Spt.TargetId = 0;
    sptwb.Spt.Lun = 0;
    sptwb.Spt.SenseInfoLength = 24;
    sptwb.Spt.DataIn = SCSI_IOCTL_DATA_IN;
    sptwb.Spt.DataTransferLength = READ_ATTRIBUTE_BUFFER_SIZE;
    sptwb.Spt.TimeOutValue = 2;
    sptwb.Spt.DataBufferOffset = offsetof(SCSI_PASS_THROUGH_WITH_BUFFERS,
DataBuf);
    sptwb.Spt.SenseInfoOffset = offsetof(SCSI_PASS_THROUGH_WITH_BUFFERS,
SenseBuf);

	COMMAND_TYPE type = asi->CommandType;
	if(type == CMD_TYPE_SAT)
	{
		sptwb.Spt.CdbLength = 12;
		sptwb.Spt.Cdb[0] = 0xA1;//ATA прохід через (12) операцій коду (A1h)
		sptwb.Spt.Cdb[1] = (4 << 1) | 0; //MULTIPLE_COUNT=0,PROTOCOL=4(PIO
Data-In),Reserved
		sptwb.Spt.Cdb[2] = (1 << 3) | (1 << 2) |
2;//OFF_LINE=0,CK_COND=0,Reserved=0,T_DIR=1(ToDevice),BYTE_BLOCK=1,T_LENGTH=2
		sptwb.Spt.Cdb[3] = READ_ATTRIBUTES;//FEATURES (7:0)
		sptwb.Spt.Cdb[4] = 1;//SECTOR_COUNT (7:0)
		sptwb.Spt.Cdb[5] = 1;//LBA_LOW (7:0)
		sptwb.Spt.Cdb[6] = SMART_CYL_LOW;//LBA_MID (7:0)
		sptwb.Spt.Cdb[7] = SMART_CYL_HI;//LBA_HIGH (7:0)
		sptwb.Spt.Cdb[9] = SMART_CMD;//COMMAND
	}
	else if(type == CMD_TYPE_SUNPLUS)
	{
		sptwb.Spt.CdbLength = 12;
		sptwb.Spt.Cdb[0] = 0xF8;
		sptwb.Spt.Cdb[1] = 0x00;
		sptwb.Spt.Cdb[2] = 0x22;
		sptwb.Spt.Cdb[3] = 0x10;
		sptwb.Spt.Cdb[4] = 0x01;
		sptwb.Spt.Cdb[5] = 0xD0;  // READ_ATTRIBUTES
```

```
        sptwb.Spt.Cdb[6] = 0x01;
        sptwb.Spt.Cdb[7] = 0x00;
        sptwb.Spt.Cdb[8] = 0x4F;  // SMART_CYL_LOW
        sptwb.Spt.Cdb[9] = 0xC2;  // SMART_CYL_HIGH
        sptwb.Spt.Cdb[10] = 0xA0;
        sptwb.Spt.Cdb[11] = 0xB0; // SMART_CMD
    }
    else if(type == CMD_TYPE_IO_DATA)
    {
        sptwb.Spt.CdbLength = 12;
        sptwb.Spt.Cdb[0] = 0xE3;
        sptwb.Spt.Cdb[1] = 0x00; // ?
        sptwb.Spt.Cdb[2] = 0xD0; // READ_ATTRIBUTES
        sptwb.Spt.Cdb[3] = 0x00; // ?
        sptwb.Spt.Cdb[4] = 0x00; // ?
        sptwb.Spt.Cdb[5] = 0x4F; // SMART_CYL_LOW
        sptwb.Spt.Cdb[6] = 0xC2; // SMART_CYL_HIGH
        sptwb.Spt.Cdb[7] = 0xA0; //
        sptwb.Spt.Cdb[8] = 0xB0; // SMART_CMD
        sptwb.Spt.Cdb[9] = 0x00;
        sptwb.Spt.Cdb[10] = 0x00;
        sptwb.Spt.Cdb[11] = 0x00;
    }
    else if(type == CMD_TYPE_LOGITEC)
    {
        sptwb.Spt.CdbLength = 10;
        sptwb.Spt.Cdb[0] = 0xE0;
        sptwb.Spt.Cdb[1] = 0x00;
        sptwb.Spt.Cdb[2] = 0xD0; // READ_ATTRIBUTES
        sptwb.Spt.Cdb[3] = 0x00;
        sptwb.Spt.Cdb[4] = 0x00;
        sptwb.Spt.Cdb[5] = 0x4F; // SMART_CYL_LOW
        sptwb.Spt.Cdb[6] = 0xC2; // SMART_CYL_HIGH
        sptwb.Spt.Cdb[7] = 0xA0;
        sptwb.Spt.Cdb[8] = 0xB0; // SMART_CMD
        sptwb.Spt.Cdb[9] = 0x4C;
    }
    else if(type == CMD_TYPE_JMICRON)
    {
        sptwb.Spt.CdbLength = 12;
        sptwb.Spt.Cdb[0] = 0xDF;
        sptwb.Spt.Cdb[1] = 0x10;
        sptwb.Spt.Cdb[2] = 0x00;
        sptwb.Spt.Cdb[3] = 0x02;
        sptwb.Spt.Cdb[4] = 0x00;
        sptwb.Spt.Cdb[5] = 0xD0;  // READ_ATTRIBUTES
        sptwb.Spt.Cdb[6] = 0x01;
        sptwb.Spt.Cdb[7] = 0x01;
        sptwb.Spt.Cdb[8] = 0x4F;  // SMART_CYL_LOW
        sptwb.Spt.Cdb[9] = 0xC2;  // SMART_CYL_HIGH
        sptwb.Spt.Cdb[10] = 0xA0;
        sptwb.Spt.Cdb[11] = 0xB0; // SMART_CMD
    }
    else if(type == CMD_TYPE_CYPRESS)
    {
        sptwb.Spt.CdbLength = 16;
        sptwb.Spt.Cdb[0] = 0x24;
        sptwb.Spt.Cdb[1] = 0x24;
        sptwb.Spt.Cdb[2] = 0x00;
        sptwb.Spt.Cdb[3] = 0xBE;
        sptwb.Spt.Cdb[4] = 0x01;
        sptwb.Spt.Cdb[5] = 0x00;
        sptwb.Spt.Cdb[6] = 0xD0;  // READ_ATTRIBUTES
        sptwb.Spt.Cdb[7] = 0x00;
        sptwb.Spt.Cdb[8] = 0x00;
        sptwb.Spt.Cdb[9] = 0x4F;  // SMART_CYL_LOW
        sptwb.Spt.Cdb[10] = 0xC2; // SMART_CYL_HIGH
        sptwb.Spt.Cdb[11] = 0xA0;
        sptwb.Spt.Cdb[12] = 0xB0; // ID_CMD
```

```
            sptwb.Spt.Cdb[13] = 0x00;
            sptwb.Spt.Cdb[14] = 0x00;
            sptwb.Spt.Cdb[15] = 0x00;
      }
      else
      {
            return FALSE;
      }

      length = offsetof(SCSI_PASS_THROUGH_WITH_BUFFERS, DataBuf) +
sptwb.Spt.DataTransferLength;
      bRet = ::DeviceIoControl(hIoCtrl, IOCTL_SCSI_PASS_THROUGH,
            &sptwb, sizeof(SCSI_PASS_THROUGH),
            &sptwb, length,   &dwReturned, NULL);

      ::CloseHandle(hIoCtrl);

      if(bRet == FALSE || dwReturned != length)
      {
            return      FALSE;
      }

      CString str;
      asi->AttributeCount = 0;
      int j = 0;
      for(int i = 0; i < MAX_ATTRIBUTE; i++)
      {
            DWORD rawValue = 0;
            memcpy(     &(asi->Attribute[j]),
                        &(sptwb.DataBuf[i * sizeof(SMART_ATTRIBUTE) + 2]),
sizeof(SMART_ATTRIBUTE));

            if(asi->Attribute[j].Id != 0)
            {
                  switch(asi->Attribute[j].Id)
                  {
                  case 0x09: // Кількість відпрацьованих годин у включеному
стані
                        rawValue = MAKELONG(
                              MAKEWORD(asi->Attribute[j].RawValue[0], asi-
>Attribute[j].RawValue[1]),
                              MAKEWORD(asi->Attribute[j].RawValue[2], asi-
>Attribute[j].RawValue[3])
                              );
                        asi->PowerOnRawValue = rawValue;
                        asi->DetectedPowerOnHours = GetPowerOnHours(rawValue,
asi->DetectedTimeUnitType);
                        asi->MeasuredPowerOnHours = GetPowerOnHours(rawValue,
asi->MeasuredTimeUnitType);
                        break;
                  case 0x0C: // Кількість зафіксованих повторів
включення/вимикання живлення накопичувача
                        rawValue = MAKELONG(
                              MAKEWORD(asi->Attribute[j].RawValue[0], asi-
>Attribute[j].RawValue[1]),
                              MAKEWORD(asi->Attribute[j].RawValue[2], asi-
>Attribute[j].RawValue[3])
                              );
                        asi->PowerOnCount = rawValue;
                        break;
                  case 0xC2: // Температура
                        if(asi->Attribute[j].RawValue[0] > 0)
                        {
                              asi->Temperature = asi->Attribute[j].RawValue[0];
                        }
                        else
                        {
                              asi->Temperature = asi->Attribute[j].CurrentValue;
                        }
```

```
                        break;
                  case 0xBB: // Визначення фірми-розробника
                        if(asi->VendorId == VENDOR_MTRON)
                        {
                              asi->Life = asi->Attribute[j].CurrentValue;
                        // РОЗШИФРУВАННЯ ЗНАЧЕНЬ
                        //    asi->Life = 0;
                        //    asi->Attribute[j].CurrentValue = 0;
                        }
                        break;
                  default:
                        break;
                  }
                  j++;
            }
      }
      asi->AttributeCount = j;

      if(asi->AttributeCount > 0)
      {
            return TRUE;
      }
      else
      {
            return FALSE;
      }
}

BOOL CAtaSmart::GetSmartThresholdSat(INT physicalDriveId, ATA_SMART_INFO* asi)
{
      BOOL  bRet;
      HANDLE      hIoCtrl;
      DWORD dwReturned;
      DWORD length;

      SCSI_PASS_THROUGH_WITH_BUFFERS sptwb;

      hIoCtrl = GetIoCtrlHandle(physicalDriveId);
      if(hIoCtrl == INVALID_HANDLE_VALUE)
      {
            return      FALSE;
      }

      ::ZeroMemory(&sptwb,sizeof(SCSI_PASS_THROUGH_WITH_BUFFERS));

    sptwb.Spt.Length = sizeof(SCSI_PASS_THROUGH);
    sptwb.Spt.PathId = 0;
    sptwb.Spt.TargetId = 0;
    sptwb.Spt.Lun = 0;
    sptwb.Spt.SenseInfoLength = 24;
    sptwb.Spt.DataIn = SCSI_IOCTL_DATA_IN;
    sptwb.Spt.DataTransferLength = READ_THRESHOLD_BUFFER_SIZE;
    sptwb.Spt.TimeOutValue = 2;
    sptwb.Spt.DataBufferOffset = offsetof(SCSI_PASS_THROUGH_WITH_BUFFERS,
DataBuf);
    sptwb.Spt.SenseInfoOffset = offsetof(SCSI_PASS_THROUGH_WITH_BUFFERS,
SenseBuf);

      COMMAND_TYPE type = asi->CommandType;
      if(type == CMD_TYPE_SAT)
      {
            sptwb.Spt.CdbLength = 12;
            sptwb.Spt.Cdb[0] = 0xA1; ////ATA Проходів через (12) операцій коду
(A1h)
            sptwb.Spt.Cdb[1] = (4 << 1) | 0; //MULTIPLE_COUNT=0,PROTOCOL=4(PIO
Data-In),Reserved
            sptwb.Spt.Cdb[2] = (1 << 3) | (1 << 2) |
2;//OFF_LINE=0,CK_COND=0,Reserved=0,T_DIR=1(ToDevice),BYTE_BLOCK=1,T_LENGTH=2
            sptwb.Spt.Cdb[3] = READ_THRESHOLDS;//FEATURES (7:0)
```

```
        sptwb.Spt.Cdb[4] = 1;//SECTOR_COUNT (7:0)
        sptwb.Spt.Cdb[5] = 1;//LBA_LOW (7:0)
        sptwb.Spt.Cdb[6] = SMART_CYL_LOW;//LBA_MID (7:0)
        sptwb.Spt.Cdb[7] = SMART_CYL_HI;//LBA_HIGH (7:0)
        sptwb.Spt.Cdb[9] = SMART_CMD;//COMMAND
    }
    else if(type == CMD_TYPE_SUNPLUS)
    {
        sptwb.Spt.CdbLength = 12;
        sptwb.Spt.Cdb[0] = 0xF8;
        sptwb.Spt.Cdb[1] = 0x00;
        sptwb.Spt.Cdb[2] = 0x22;
        sptwb.Spt.Cdb[3] = 0x10;
        sptwb.Spt.Cdb[4] = 0x01;
        sptwb.Spt.Cdb[5] = 0xD1; // READ_THRESHOLD
        sptwb.Spt.Cdb[6] = 0x01;
        sptwb.Spt.Cdb[7] = 0x01;
        sptwb.Spt.Cdb[8] = 0x4F; // SMART_CYL_LOW
        sptwb.Spt.Cdb[9] = 0xC2; // SMART_CYL_HIGH
        sptwb.Spt.Cdb[10] = 0xA0;
        sptwb.Spt.Cdb[11] = 0xB0;// SMART_CMD
    }
    else if(type == CMD_TYPE_IO_DATA)
    {
        sptwb.Spt.CdbLength = 12;
        sptwb.Spt.Cdb[0] = 0xE3;
        sptwb.Spt.Cdb[1] = 0x00; // ?
        sptwb.Spt.Cdb[2] = 0xD1; // READ_THRESHOLD
        sptwb.Spt.Cdb[3] = 0x00; // ?
        sptwb.Spt.Cdb[4] = 0x00; // ?
        sptwb.Spt.Cdb[5] = 0x4F; // SMART_CYL_LOW
        sptwb.Spt.Cdb[6] = 0xC2; // SMART_CYL_HIGH
        sptwb.Spt.Cdb[7] = 0xA0; //
        sptwb.Spt.Cdb[8] = 0xB0; // SMART_CMD
        sptwb.Spt.Cdb[9] = 0x00;
        sptwb.Spt.Cdb[10] = 0x00;
        sptwb.Spt.Cdb[11] = 0x00;
    }
    else if(type == CMD_TYPE_LOGITEC)
    {
        sptwb.Spt.CdbLength = 10;
        sptwb.Spt.Cdb[0] = 0xE0;
        sptwb.Spt.Cdb[1] = 0x00;
        sptwb.Spt.Cdb[2] = 0xD1; // READ_THRESHOLD
        sptwb.Spt.Cdb[3] = 0x00;
        sptwb.Spt.Cdb[4] = 0x00;
        sptwb.Spt.Cdb[5] = 0x4F; // SMART_CYL_LOW
        sptwb.Spt.Cdb[6] = 0xC2; // SMART_CYL_HIGH
        sptwb.Spt.Cdb[7] = 0xA0;
        sptwb.Spt.Cdb[8] = 0xB0; // SMART_CMD
        sptwb.Spt.Cdb[9] = 0x4C;
    }
    else if(type == CMD_TYPE_JMICRON)
    {
        sptwb.Spt.CdbLength = 12;
        sptwb.Spt.Cdb[0] = 0xDF;
        sptwb.Spt.Cdb[1] = 0x10;
        sptwb.Spt.Cdb[2] = 0x00;
        sptwb.Spt.Cdb[3] = 0x02;
        sptwb.Spt.Cdb[4] = 0x00;
        sptwb.Spt.Cdb[5] = 0xD1;  // READ_THRESHOLD
        sptwb.Spt.Cdb[6] = 0x01;
        sptwb.Spt.Cdb[7] = 0x01;
        sptwb.Spt.Cdb[8] = 0x4F;  // SMART_CYL_LOW
        sptwb.Spt.Cdb[9] = 0xC2;  // SMART_CYL_HIGH
        sptwb.Spt.Cdb[10] = 0xA0;
        sptwb.Spt.Cdb[11] = 0xB0; // SMART_CMD
    }
    else if(type == CMD_TYPE_CYPRESS)
```

```
	{
		sptwb.Spt.CdbLength = 16;
		sptwb.Spt.Cdb[0] = 0x24;
		sptwb.Spt.Cdb[1] = 0x24;
		sptwb.Spt.Cdb[2] = 0x00;
		sptwb.Spt.Cdb[3] = 0xBE;
		sptwb.Spt.Cdb[4] = 0x01;
		sptwb.Spt.Cdb[5] = 0x00;
		sptwb.Spt.Cdb[6] = 0xD1;  // READ_THRESHOLD
		sptwb.Spt.Cdb[7] = 0x00;
		sptwb.Spt.Cdb[8] = 0x00;
		sptwb.Spt.Cdb[9] = 0x4F;  // SMART_CYL_LOW
		sptwb.Spt.Cdb[10] = 0xC2; // SMART_CYL_HIGH
		sptwb.Spt.Cdb[11] = 0xA0;
		sptwb.Spt.Cdb[12] = 0xB0; // ID_CMD
		sptwb.Spt.Cdb[13] = 0x00;
		sptwb.Spt.Cdb[14] = 0x00;
		sptwb.Spt.Cdb[15] = 0x00;
	}
	else
	{
		return FALSE;
	}

	length = offsetof(SCSI_PASS_THROUGH_WITH_BUFFERS, DataBuf) +
sptwb.Spt.DataTransferLength;
	bRet = ::DeviceIoControl(hIoCtrl, IOCTL_SCSI_PASS_THROUGH,
		&sptwb, sizeof(SCSI_PASS_THROUGH),
		&sptwb, length,   &dwReturned, NULL);

	::CloseHandle(hIoCtrl);

	if(bRet == FALSE || dwReturned != length)
	{
		return	FALSE;
	}

	CString str;
	int j = 0;
	for(int i = 0; i < MAX_ATTRIBUTE; i++)
	{
		memcpy(	&(asi->Threshold[i]),
			&(sptwb.DataBuf[i * sizeof(SMART_THRESHOLD) + 2]),
sizeof(SMART_THRESHOLD));

		if(asi->Threshold[j].Id != 0)
		{
			j++;
		}
	}

	return	TRUE;
}

BOOL CAtaSmart::ControlSmartStatusSat(INT physicalDriveId, BYTE command,
COMMAND_TYPE type)
{
	BOOL  bRet;
	HANDLE	hIoCtrl;
	DWORD dwReturned;

	SCSI_PASS_THROUGH_WITH_BUFFERS sptwb;

	hIoCtrl = GetIoCtrlHandle(physicalDriveId);
	if(hIoCtrl == INVALID_HANDLE_VALUE)
	{
		return	FALSE;
	}
```

```
      ::ZeroMemory(&sptwb,sizeof(SCSI_PASS_THROUGH_WITH_BUFFERS));

    sptwb.Spt.Length = sizeof(SCSI_PASS_THROUGH);
    sptwb.Spt.PathId = 0;
    sptwb.Spt.TargetId = 0;
    sptwb.Spt.Lun = 0;
    sptwb.Spt.SenseInfoLength = 24;
    sptwb.Spt.DataIn = SCSI_IOCTL_DATA_IN;
    sptwb.Spt.DataTransferLength = 0;
    sptwb.Spt.TimeOutValue = 2;
    sptwb.Spt.DataBufferOffset = offsetof(SCSI_PASS_THROUGH_WITH_BUFFERS,
DataBuf);
    sptwb.Spt.SenseInfoOffset = offsetof(SCSI_PASS_THROUGH_WITH_BUFFERS,
SenseBuf);
      if(type == CMD_TYPE_SAT)
      {
            sptwb.Spt.CdbLength = 12;
            sptwb.Spt.Cdb[0] = 0xA1; //ATA Проходів через (12) операцій коду
(A1h)
            sptwb.Spt.Cdb[1] = (3 << 1) | 0; //MULTIPLE_COUNT=0,PROTOCOL=3(Non-
Data),Reserved
            sptwb.Spt.Cdb[2] = (1 << 3) | (1 << 2) |
2;//OFF_LINE=0,CK_COND=0,Reserved=0,T_DIR=1(ToDevice),BYTE_BLOCK=1,T_LENGTH=2
            sptwb.Spt.Cdb[3] = command;//FEATURES (7:0)
            sptwb.Spt.Cdb[4] = 0;//SECTOR_COUNT (7:0)
            sptwb.Spt.Cdb[5] = 1;//LBA_LOW (7:0)
            sptwb.Spt.Cdb[6] = SMART_CYL_LOW;//LBA_MID (7:0)
            sptwb.Spt.Cdb[7] = SMART_CYL_HI;//LBA_HIGH (7:0)
            sptwb.Spt.Cdb[9] = SMART_CMD;//COMMAND
      }
      else if(type == CMD_TYPE_SUNPLUS)
      {
            sptwb.Spt.CdbLength = 12;
            sptwb.Spt.Cdb[0] = 0xF8;
            sptwb.Spt.Cdb[1] = 0x00;
            sptwb.Spt.Cdb[2] = 0x22;
            sptwb.Spt.Cdb[3] = 0x10;
            sptwb.Spt.Cdb[4] = 0x01;
            sptwb.Spt.Cdb[5] = command;
            sptwb.Spt.Cdb[6] = 0x01;
            sptwb.Spt.Cdb[7] = 0x00;
            sptwb.Spt.Cdb[8] = 0x4F;  // SMART_CYL_LOW
            sptwb.Spt.Cdb[9] = 0xC2;  // SMART_CYL_HIGH
            sptwb.Spt.Cdb[10] = 0xA0;
            sptwb.Spt.Cdb[11] = 0xB0; // SMART_CMD
      }
      else if(type == CMD_TYPE_IO_DATA)
      {
            sptwb.Spt.CdbLength = 12;
            sptwb.Spt.Cdb[0] = 0xE3;
            sptwb.Spt.Cdb[1] = 0x00; // ?
            sptwb.Spt.Cdb[2] = command;
            sptwb.Spt.Cdb[3] = 0x00; // ?
            sptwb.Spt.Cdb[4] = 0x00; // ?
            sptwb.Spt.Cdb[5] = 0x4F; // SMART_CYL_LOW
            sptwb.Spt.Cdb[6] = 0xC2; // SMART_CYL_HIGH
            sptwb.Spt.Cdb[7] = 0xA0; //
            sptwb.Spt.Cdb[8] = 0xB0; // SMART_CMD
            sptwb.Spt.Cdb[9] = 0x00;
            sptwb.Spt.Cdb[10] = 0x00;
            sptwb.Spt.Cdb[11] = 0x00;
      }
      else if(type == CMD_TYPE_LOGITEC)
      {
            sptwb.Spt.CdbLength = 10;
            sptwb.Spt.Cdb[0] = 0xE0;
            sptwb.Spt.Cdb[1] = 0x00;
            sptwb.Spt.Cdb[2] = command;
            sptwb.Spt.Cdb[3] = 0x00;
```

```
		sptwb.Spt.Cdb[4] = 0x00;
		sptwb.Spt.Cdb[5] = 0x4F; // SMART_CYL_LOW
		sptwb.Spt.Cdb[6] = 0xC2; // SMART_CYL_HIGH
		sptwb.Spt.Cdb[7] = 0xA0;
		sptwb.Spt.Cdb[8] = 0xB0; // SMART_CMD
		sptwb.Spt.Cdb[9] = 0x4C;
	}
	else if(type == CMD_TYPE_JMICRON)
	{
		sptwb.Spt.CdbLength = 12;
		sptwb.Spt.Cdb[0] = 0xDF;
		sptwb.Spt.Cdb[1] = 0x10;
		sptwb.Spt.Cdb[2] = 0x00;
		sptwb.Spt.Cdb[3] = 0x02;
		sptwb.Spt.Cdb[4] = 0x00;
		sptwb.Spt.Cdb[5] = command;
		sptwb.Spt.Cdb[6] = 0x01;
		sptwb.Spt.Cdb[7] = 0x01;
		sptwb.Spt.Cdb[8] = 0x4F;  // SMART_CYL_LOW
		sptwb.Spt.Cdb[9] = 0xC2;  // SMART_CYL_HIGH
		sptwb.Spt.Cdb[10] = 0xA0;
		sptwb.Spt.Cdb[11] = 0xB0; // SMART_CMD
	}
	else if(type == CMD_TYPE_CYPRESS)
	{
		sptwb.Spt.CdbLength = 16;
		sptwb.Spt.Cdb[0] = 0x24;
		sptwb.Spt.Cdb[1] = 0x24;
		sptwb.Spt.Cdb[2] = 0x00;
		sptwb.Spt.Cdb[3] = 0xBE;
		sptwb.Spt.Cdb[4] = 0x00;
		sptwb.Spt.Cdb[5] = 0x00;
		sptwb.Spt.Cdb[6] = command;
		sptwb.Spt.Cdb[7] = 0x00;
		sptwb.Spt.Cdb[8] = 0x00;
		sptwb.Spt.Cdb[9] = 0x4F;  // SMART_CYL_LOW
		sptwb.Spt.Cdb[10] = 0xC2; // SMART_CYL_HIGH
		sptwb.Spt.Cdb[11] = 0xA0;
		sptwb.Spt.Cdb[12] = 0xB0; // ID_CMD
		sptwb.Spt.Cdb[13] = 0x00;
		sptwb.Spt.Cdb[14] = 0x00;
		sptwb.Spt.Cdb[15] = 0x00;
	}
	else
	{
		return FALSE;
	}

	DWORD length = offsetof(SCSI_PASS_THROUGH_WITH_BUFFERS, DataBuf) +
sptwb.Spt.DataTransferLength;
	bRet = ::DeviceIoControl(hIoCtrl, IOCTL_SCSI_PASS_THROUGH,
		&sptwb, sizeof(SCSI_PASS_THROUGH),
		&sptwb, length,   &dwReturned, NULL);

	::CloseHandle(hIoCtrl);

	return	bRet;
}

BOOL CAtaSmart::SendAtaCommandSat(INT physicalDriveId, BYTE main, BYTE sub, BYTE
param, COMMAND_TYPE type)
{
	BOOL  bRet;
	HANDLE	hIoCtrl;
	DWORD dwReturned;

	SCSI_PASS_THROUGH_WITH_BUFFERS sptwb;

	hIoCtrl = GetIoCtrlHandle(physicalDriveId);
```

```
	if(hIoCtrl == INVALID_HANDLE_VALUE)
	{
		return	FALSE;
	}

	::ZeroMemory(&sptwb,sizeof(SCSI_PASS_THROUGH_WITH_BUFFERS));

    sptwb.Spt.Length = sizeof(SCSI_PASS_THROUGH);
    sptwb.Spt.PathId = 0;
    sptwb.Spt.TargetId = 0;
    sptwb.Spt.Lun = 0;
    sptwb.Spt.SenseInfoLength = 24;
    sptwb.Spt.DataIn = SCSI_IOCTL_DATA_IN;
    sptwb.Spt.DataTransferLength = 0;
    sptwb.Spt.TimeOutValue = 2;
    sptwb.Spt.DataBufferOffset = offsetof(SCSI_PASS_THROUGH_WITH_BUFFERS,
DataBuf);
    sptwb.Spt.SenseInfoOffset = offsetof(SCSI_PASS_THROUGH_WITH_BUFFERS,
SenseBuf);
	if(type == CMD_TYPE_SAT)
	{
		sptwb.Spt.CdbLength = 12;
		sptwb.Spt.Cdb[0]  = 0xA1; //ATA Проходів через (12) операцій коду
(A1h)
		sptwb.Spt.Cdb[1]  = (3 << 1) | 0; //MULTIPLE_COUNT=0,PROTOCOL=3(Non-
Data),Reserved
		sptwb.Spt.Cdb[2]  = (1 << 3) | (1 << 2) |
2;//OFF_LINE=0,CK_COND=0,Reserved=0,T_DIR=1(ToDevice),BYTE_BLOCK=1,T_LENGTH=2
		sptwb.Spt.Cdb[3]  = sub;		//FEATURES (7:0)
		sptwb.Spt.Cdb[4]  = param;		//SECTOR_COUNT (7:0)
		sptwb.Spt.Cdb[5]  = 0x00;		//LBA_LOW (7:0)
		sptwb.Spt.Cdb[6]  = 0x00;		//LBA_MID (7:0)
		sptwb.Spt.Cdb[7]  = 0x00;		//LBA_HIGH (7:0)
		sptwb.Spt.Cdb[8]  = 0xA0;		//DEVICE_HEAD
		sptwb.Spt.Cdb[9]  = main;		//COMMAND
		sptwb.Spt.Cdb[10] = 0x00;
		sptwb.Spt.Cdb[11] = 0x00;
	}
	else if(type == CMD_TYPE_SUNPLUS)
	{
		sptwb.Spt.CdbLength = 12;
		sptwb.Spt.Cdb[0]  = 0xF8;
		sptwb.Spt.Cdb[1]  = 0x00;
		sptwb.Spt.Cdb[2]  = 0x22;
		sptwb.Spt.Cdb[3]  = 0x10;
		sptwb.Spt.Cdb[4]  = 0x01;
		sptwb.Spt.Cdb[5]  = sub;
		sptwb.Spt.Cdb[6]  = param;
		sptwb.Spt.Cdb[7]  = 0x00;
		sptwb.Spt.Cdb[8]  = 0x00;
		sptwb.Spt.Cdb[9]  = 0x00;
		sptwb.Spt.Cdb[10] = 0xA0;
		sptwb.Spt.Cdb[11] = main;
	}
	else if(type == CMD_TYPE_IO_DATA)
	{
		sptwb.Spt.CdbLength = 12;
		sptwb.Spt.Cdb[0]  = 0xE3;
		sptwb.Spt.Cdb[1]  = 0x00;
		sptwb.Spt.Cdb[2]  = sub;
		sptwb.Spt.Cdb[3]  = param;
		sptwb.Spt.Cdb[4]  = 0x00;
		sptwb.Spt.Cdb[5]  = 0x00;
		sptwb.Spt.Cdb[6]  = 0x00;
		sptwb.Spt.Cdb[7]  = 0xA0;
		sptwb.Spt.Cdb[8]  = main;
		sptwb.Spt.Cdb[9]  = 0x00;
		sptwb.Spt.Cdb[10] = 0x00;
		sptwb.Spt.Cdb[11] = 0x00;
```

```
	}
	else if(type == CMD_TYPE_LOGITEC)
	{
		sptwb.Spt.CdbLength = 10;
		sptwb.Spt.Cdb[0] = 0xE0;
		sptwb.Spt.Cdb[1] = 0x00;
		sptwb.Spt.Cdb[2] = sub;
		sptwb.Spt.Cdb[3] = param;
		sptwb.Spt.Cdb[4] = 0x00;
		sptwb.Spt.Cdb[5] = 0x00;
		sptwb.Spt.Cdb[6] = 0x00;
		sptwb.Spt.Cdb[7] = 0xA0;
		sptwb.Spt.Cdb[8] = main;
		sptwb.Spt.Cdb[9] = 0x4C;		// ?
	}
	else if(type == CMD_TYPE_JMICRON)
	{
		sptwb.Spt.CdbLength = 12;
		sptwb.Spt.Cdb[0]  = 0xDF;
		sptwb.Spt.Cdb[1]  = 0x10;
		sptwb.Spt.Cdb[2]  = 0x00;
		sptwb.Spt.Cdb[3]  = 0x02;
		sptwb.Spt.Cdb[4]  = 0x00;
		sptwb.Spt.Cdb[5]  = sub;
		sptwb.Spt.Cdb[6]  = param;
		sptwb.Spt.Cdb[7]  = 0x00;
		sptwb.Spt.Cdb[8]  = 0x00;
		sptwb.Spt.Cdb[9]  = 0x00;
		sptwb.Spt.Cdb[10] = 0xA0;
		sptwb.Spt.Cdb[11] = main;
	}
	else if(type == CMD_TYPE_CYPRESS)
	{
		sptwb.Spt.CdbLength = 16;
		sptwb.Spt.Cdb[0]  = 0x24;
		sptwb.Spt.Cdb[1]  = 0x24;
		sptwb.Spt.Cdb[2]  = 0x00;
		sptwb.Spt.Cdb[3]  = 0xBE;
		sptwb.Spt.Cdb[4]  = 0x00;
		sptwb.Spt.Cdb[5]  = 0x00;
		sptwb.Spt.Cdb[6]  = sub;
		sptwb.Spt.Cdb[7]  = param;
		sptwb.Spt.Cdb[8]  = 0x00;
		sptwb.Spt.Cdb[9]  = 0x00;
		sptwb.Spt.Cdb[10] = 0x00;
		sptwb.Spt.Cdb[11] = 0xA0;
		sptwb.Spt.Cdb[12] = main;
		sptwb.Spt.Cdb[13] = 0x00;
		sptwb.Spt.Cdb[14] = 0x00;
		sptwb.Spt.Cdb[15] = 0x00;
	}
	else
	{
		return FALSE;
	}

	DWORD length = offsetof(SCSI_PASS_THROUGH_WITH_BUFFERS, DataBuf) +
sptwb.Spt.DataTransferLength;
	bRet = ::DeviceIoControl(hIoCtrl, IOCTL_SCSI_PASS_THROUGH,
		&sptwb, sizeof(SCSI_PASS_THROUGH),
		&sptwb, length,   &dwReturned, NULL);

	::CloseHandle(hIoCtrl);

	return	bRet;
}

/*-------------------------------------------------------------------------*/
// Підтримка функцій
```

```
/*-----------------------------------------------------------------------------*/

DWORD CAtaSmart::CheckDiskStatus(SMART_ATTRIBUTE* attribute, SMART_THRESHOLD*
threshold, DWORD attributeCount, DWORD vendorId)
{
	int error = 0;
	int caution = 0;
	BOOL flagUnknown = TRUE;

	for(DWORD j = 0; j < attributeCount; j++)
	{
		if( attribute[j].Id != 0xBE // Температура воздуха
		&&	threshold[j].ThresholdValue != 0
		&&	attribute[j].CurrentValue <= threshold[j].ThresholdValue)
		{
			error++;
		}

		switch(attribute[j].Id)
		{
		case 0x05: // Кількість перепризначених секторів
//		case 0xC4: // Кількість операцій перепризначення (ремаппінгу).
		case 0xC5: // Поточна кількість нестабільних секторів
		case 0xC6: // Кількість нескоректованих помилок
			if(attribute[j].RawValue[0] == 0xFF
			&& attribute[j].RawValue[1] == 0xFF
			&& attribute[j].RawValue[2] == 0xFF
			&& attribute[j].RawValue[3] == 0xFF)
			{
				flagUnknown = FALSE;
			}
			else
			{
				caution += attribute[j].RawValue[0]
						+ attribute[j].RawValue[1];
				flagUnknown = FALSE;
			}
			break;
		case 0xBB: // Визначення фірми-розробника
			if(vendorId == VENDOR_MTRON)
			{
				if(attribute[j].CurrentValue == 0)
				{
					error = 1;
				}
				else if(attribute[j].CurrentValue < 10)
				{
					caution = 1;
				}
				else
				{
					flagUnknown = FALSE;
				}
			}
			break;
		default:
			break;
		}
	}

	if(error > 0)
	{
		return DISK_STATUS_BAD;
	}
	else if(flagUnknown)
	{
		return DISK_STATUS_UNKNOWN;
	}
	else if(caution > 0)
```

```
	{
		return DISK_STATUS_CAUTION;
	}
	else
	{
		return DISK_STATUS_GOOD;
	}
}

VOID CAtaSmart::ChangeByteOrder(PCHAR str, DWORD length)
{
	CHAR  temp;
	for(DWORD i = 0; i < length; i += 2)
	{
		temp = str[i];
		str[i] = str[i+1];
		str[i+1] = temp;
	}
}

BOOL CAtaSmart::CheckAsciiStringError(PCHAR str, DWORD length)
{
	BOOL flag = FALSE;
	for(DWORD i = 0; i < length; i++)
	{
		if((0x00 < str[i] && str[i] < 0x20) || str[i] >= 0x7f)
		{
			flag = TRUE;
			break;
		}
	}
	return flag;
}

DWORD CAtaSmart::GetPowerOnHours(DWORD rawValue, DWORD timeUnitType)
{
	switch(timeUnitType)
	{
	case POWER_ON_UNKNOWN:
		return 0;
		break;
	case POWER_ON_HOURS:
		return rawValue;
		break;
	case POWER_ON_MINUTES:
		return rawValue / 60;
		break;
	case POWER_ON_HALF_MINUTES:
		return rawValue / 120;
		break;
	case POWER_ON_SECONDS:
		return rawValue / 60 / 60;
		break;
	default:
		return rawValue;
		break;
	}
}

DWORD CAtaSmart::GetPowerOnHoursEx(DWORD i, DWORD timeUnitType)
{
	DWORD rawValue = vars[i].PowerOnRawValue;
	switch(timeUnitType)
	{
	case POWER_ON_UNKNOWN:
		return 0;
		break;
	case POWER_ON_HOURS:
		return rawValue;
```

```
            break;
      case POWER_ON_MINUTES:
            return rawValue / 60;
            break;
      case POWER_ON_HALF_MINUTES:
            return rawValue / 120;
            break;
      case POWER_ON_SECONDS:
            return rawValue / 60 / 60;
            break;
      default:
            return rawValue;
            break;
      }
}

DWORD CAtaSmart::GetTransferMode(WORD w63, WORD w76, WORD w88, CString &current,
CString &max, CString &type, INTERFACE_TYPE* interfaceType)
{
      DWORD tm = TRANSFER_MODE_PIO;
      current = max = _T("");
      type = _T("Parallel ATA");
      *interfaceType = INTERFACE_TYPE_PATA;

      // Слово DMA або PIO
      if(w63 & 0x0700)
      {
            tm = TRANSFER_MODE_PIO_DMA;
            current = max = _T("PIO / DMA");
      }

      // Ultara DMA Максимальний режим передачі
             if(w88 & 0x0040){tm = TRANSFER_MODE_ULTRA_DMA_133; max = _T("Ultra
DMA/133");}
      else if(w88 & 0x0020){tm = TRANSFER_MODE_ULTRA_DMA_100; max = _T("Ultra
DMA/100");}
      else if(w88 & 0x0010){tm = TRANSFER_MODE_ULTRA_DMA_66;  max = _T("Ultra
DMA/66");}
      else if(w88 & 0x0008){tm = TRANSFER_MODE_ULTRA_DMA_44;  max = _T("Ultra
DMA/44");}
      else if(w88 & 0x0004){tm = TRANSFER_MODE_ULTRA_DMA_33;  max = _T("Ultra
DMA/33");}
      else if(w88 & 0x0002){tm = TRANSFER_MODE_ULTRA_DMA_25;  max = _T("Ultra
DMA/25");}
      else if(w88 & 0x0001){tm = TRANSFER_MODE_ULTRA_DMA_16;  max = _T("Ultra
DMA/16");}

      // Ultara DMA поточний режим передачі
             if(w88 & 0x4000){current = _T("Ultra DMA/133");}
      else if(w88 & 0x2000){current = _T("Ultra DMA/100");}
      else if(w88 & 0x1000){current = _T("Ultra DMA/66");}
      else if(w88 & 0x0800){current = _T("Ultra DMA/44");}
      else if(w88 & 0x0400){current = _T("Ultra DMA/33");}
      else if(w88 & 0x0200){current = _T("Ultra DMA/25");}
      else if(w88 & 0x0100){current = _T("Ultra DMA/16");}

      // Serial ATA
      if(w76 != 0x0000 && w76 != 0xFFFF)
      {
            current = max = _T("SATA/150");
            type = _T("Serial ATA");
            *interfaceType = INTERFACE_TYPE_SATA;
      }

             if(w76 & 0x0010){tm = TRANSFER_MODE_UNKNOWN;  current = max =
_T("Unknown");}
      else if(w76 & 0x0008){tm = TRANSFER_MODE_SATA_600; current = max =
_T("SATA/600");}
```

```
	else if(w76 & 0x0004){tm = TRANSFER_MODE_SATA_300; current = max =
_T("SATA/300");}
	else if(w76 & 0x0002){tm = TRANSFER_MODE_SATA_150; current = max =
_T("SATA/150");}

	return tm;
}

DWORD CAtaSmart::GetTimeUnitType(CString model, CString firmware, DWORD major,
DWORD transferMode)
{
	model.MakeUpper();

	if(model.Find(_T("FUJITSU")) == 0)
	{
		if(major >= 8)
		{
			return POWER_ON_HOURS;
		}
		else
		{
			return POWER_ON_SECONDS;
		}
	}
	else if(model.Find(_T("HITACHI_DK")) == 0)
	{
		return POWER_ON_MINUTES;
	}
	else if(model.Find(_T("MAXTOR")) == 0)
	{
		if(transferMode >= TRANSFER_MODE_SATA_300
		|| model.Find(_T("MAXTOR 6H")) == 0	// Maxtor DiamondMax 11
сімейство
		|| model.Find(_T("MAXTOR 7H500")) == 0	// Maxtor MaXLine Pro 500
сімейство
		|| model.Find(_T("MAXTOR 6L0")) == 0	// Maxtor DiamondMax Plus
D740X сімейство
		|| model.Find(_T("MAXTOR 4K")) == 0	// Maxtor DiamondMax D540X-
4K сімейство
		)
		{
			return POWER_ON_HOURS;
		}
		else
		{
			return POWER_ON_MINUTES;
		}
	}
	else if(model.Find(_T("SAMSUNG")) == 0)
	{
		if(transferMode >= TRANSFER_MODE_SATA_300)
		{
			return POWER_ON_HOURS;
		}
		else if(-23 >= _tstoi(firmware.Right(3)) &&
_tstoi(firmware.Right(3)) >= -39)
		{
			return POWER_ON_HALF_MINUTES;
		}
		else if(model.Find(_T("SAMSUNG SV")) == 0
		||		model.Find(_T("SAMSUNG SP")) == 0
		||		model.Find(_T("SAMSUNG HM")) == 0
		)
		{
			return POWER_ON_HALF_MINUTES;
		}
		else
		{
			return POWER_ON_HOURS;
```

```
		}
	}
	else
	{
		return POWER_ON_HOURS;
	}
}

DWORD CAtaSmart::GetAtaMajorVersion(WORD w80, CString &majorVersion)
{
	DWORD major = 0;

	if(w80 == 0x0000 || w80 == 0xFFFF)
	{
		return FALSE;
	}

	for(int i = 14; i > 0; i--)
	{
		if((w80 >> i) & 0x1)
		{
			major = i;
			break;
		}
	}

	if(major == 15)
	{
		majorVersion = _T("");
	}
	else if(major >= 8)
	{
		majorVersion.Format(_T("ATA%d-ACS"), major);
	}
	else if(major >= 4)
	{
		majorVersion.Format(_T("ATA/ATAPI-%d"), major);
	}
	else if(major == 0)
	{
		majorVersion = _T("");
	}
	else
	{
		majorVersion.Format(_T("ATA-%d"), major);
	}

	return major;
}


/*
DWORD CAtaSmart::GetMaxtorPowerOnHours(DWORD currentValue, DWORD rawValue)
{
	if(200 < currentValue && currentValue <= 253)
	{
		return ((253 - currentValue) * 65536 + rawValue) / 60;
	}
	else if(100 < currentValue && currentValue <= 200)
	{
		return ((200 - currentValue) * 65536 + rawValue) / 60;
	}
	else if(currentValue <= 100)
	{
		return ((100 - currentValue) * 65536 + rawValue) / 60;
	}
	else
	{
		return rawValue / 60;
```

```
        }
    }
    */
```

Файл AtaSmart.h – бібліотека для файлу AtaSmart.cpp

```
#pragma once

#include "winioctl.h"
#include "SPTIUtil.h"


class CAtaSmart
{
public:
	static const int MAX_DISK = 32;// FIX
	static const int MAX_ATTRIBUTE = 30; // FIX
	static const int MAX_SEARCH_PHYSICAL_DRIVE = 32;
	static const int MAX_SEARCH_SCSI_PORT = 16;
	static const int MAX_SEARCH_SCSI_TARGET_ID = 8;

	static const int SCSI_MINIPORT_BUFFER_SIZE = 512;

public:
	CAtaSmart();
	virtual ~CAtaSmart();

	enum SMART_STATUS
	{
		SMART_STATUS_NO_CHANGE = 0,
		SMART_STATUS_MINOR_CHANGE,
		SMART_STATUS_MAJOR_CHANGE
	};

	enum TRANSFER_MODE
	{
		TRANSFER_MODE_UNKNOWN = 0,
		TRANSFER_MODE_PIO,
		TRANSFER_MODE_PIO_DMA,
		TRANSFER_MODE_ULTRA_DMA_16,
		TRANSFER_MODE_ULTRA_DMA_25,
		TRANSFER_MODE_ULTRA_DMA_33,
		TRANSFER_MODE_ULTRA_DMA_44,
		TRANSFER_MODE_ULTRA_DMA_66,
		TRANSFER_MODE_ULTRA_DMA_100,
		TRANSFER_MODE_ULTRA_DMA_133,
		TRANSFER_MODE_SATA_150,
		TRANSFER_MODE_SATA_300,
		TRANSFER_MODE_SATA_600
	};

	enum DISK_STATUS
	{
		DISK_STATUS_UNKNOWN = 0,
		DISK_STATUS_GOOD,
		DISK_STATUS_CAUTION,
		DISK_STATUS_BAD
	};

	enum POWER_ON_HOURS_UNIT
	{
		POWER_ON_UNKNOWN = 0,
		POWER_ON_HOURS,
		POWER_ON_MINUTES,
		POWER_ON_HALF_MINUTES,
		POWER_ON_SECONDS,
	};

	enum COMMAND_TYPE
	{
		CMD_TYPE_PHYSICAL_DRIVE = 0,
```

```
            CMD_TYPE_SCSI_MINIPORT,
            CMD_TYPE_SAT,                        // SAT = SCSI_ATA_TRANSLATION
            CMD_TYPE_SUNPLUS,
            CMD_TYPE_IO_DATA,
            CMD_TYPE_LOGITEC,
            CMD_TYPE_JMICRON,
            CMD_TYPE_CYPRESS,
            CMD_TYPE_PROLIFIC,
            CMD_TYPE_DEBUG
      };

      enum VENDOR_ID
      {
            VENDOR_UNKNOWN      = 0x0000,
            VENDOR_MTRON    = 0x0001,

            USB_VENDOR_BUFFALO  = 0x0411,
            USB_VENDOR_IO_DATA  = 0x04BB,
            USB_VENDOR_LOGITEC  = 0x0789,
            USB_VENDOR_INITIO   = 0x13FD,
            USB_VENDOR_SUNPLUS  = 0x04FC,
            USB_VENDOR_JMICRON  = 0x152D,
            USB_VENDOR_CYPRESS  = 0x04B4,
            USB_VENDOR_OXFORD   = 0x0928,
            USB_VENDOR_PROLIFIC = 0x067B,

      };

      enum INTERFACE_TYPE
      {
            INTERFACE_TYPE_UNKNOWN = 0,
            INTERFACE_TYPE_PATA,
            INTERFACE_TYPE_SATA,
            INTERFACE_TYPE_USB,
            INTERFACE_TYPE_IEEE1394
      };

protected:
      enum IO_CONTROL_CODE
      {
            DFP_SEND_DRIVE_COMMAND  = 0x0007C084,
            DFP_RECEIVE_DRIVE_DATA  = 0x0007C088,
            IOCTL_SCSI_MINIPORT     = 0x0004D008,
            IOCTL_IDE_PASS_THROUGH  = 0x0004D028, // 2000 або пізніша версія
            IOCTL_ATA_PASS_THROUGH  = 0x0004D02C, // XP SP2 and 2003 або пізніша
версія
      };

#pragma pack(push,1)

      typedef     struct _IDENTIFY_DEVICE_OUTDATA
      {
            SENDCMDOUTPARAMS  SendCmdOutParam;
            BYTE                    Data[IDENTIFY_BUFFER_SIZE - 1];
      } IDENTIFY_DEVICE_OUTDATA, *PIDENTIFY_DEVICE_OUTDATA;

      typedef     struct _SMART_READ_DATA_OUTDATA
      {
            SENDCMDOUTPARAMS  SendCmdOutParam;
            BYTE                    Data[READ_ATTRIBUTE_BUFFER_SIZE - 1];
      } SMART_READ_DATA_OUTDATA, *PSMART_READ_DATA_OUTDATA;

      typedef struct _CMD_IDE_PATH_THROUGH
      {
            IDEREGS     reg;
            DWORD   length;
            BYTE    buffer[1];
      } CMD_IDE_PATH_THROUGH, *PCMD_IDE_PATH_THROUGH;
```

```
static const int ATA_FLAGS_DRDY_REQUIRED = 0x01;
static const int ATA_FLAGS_DATA_IN       = 0x02;
static const int ATA_FLAGS_DATA_OUT      = 0x04;
static const int ATA_FLAGS_48BIT_COMMAND = 0x08;

typedef struct _ATA_PASS_THROUGH_EX
{
      WORD    Length;
      WORD    AtaFlags;
      BYTE    PathId;
      BYTE    TargetId;
      BYTE    Lun;
      BYTE    ReservedAsUchar;
      DWORD   DataTransferLength;
      DWORD   TimeOutValue;
      DWORD   ReservedAsUlong;
      DWORD_PTR   DataBufferOffset;
      IDEREGS PreviousTaskFile;
      IDEREGS CurrentTaskFile;
} ATA_PASS_THROUGH_EX, *PCMD_ATA_PASS_THROUGH_EX;

typedef struct
{
      ATA_PASS_THROUGH_EX Apt;
      BYTE  Buf[512];
} ATA_PASS_THROUGH_EX_WITH_BUFFERS;

typedef     struct SMART_ATTRIBUTE
{
      BYTE  Id;
      WORD  StatusFlags;
      BYTE  CurrentValue;
      BYTE  WorstValue;
      BYTE  RawValue[6];
      BYTE  Reserved;
};

typedef     struct SMART_THRESHOLD
{
      BYTE  Id;
      BYTE  ThresholdValue;
      BYTE  Reserved[10];
};

typedef struct SRB_IO_CONTROL
{
   ULONG    HeaderLength;
   UCHAR    Signature[8];
   ULONG    Timeout;
   ULONG    ControlCode;
   ULONG    ReturnCode;
   ULONG    Length;
};

typedef struct SRB_IO_COMMAND
{
      SRB_IO_CONTROL    Cntrol;
      IDEREGS                 IdeRegs;
      BYTE              Data[512];
};

struct IDENTIFY_DEVICE
{
      WORD        GeneralConfiguration;                   //0
      WORD        LogicalCylinders;                       //1
Застарівший
      WORD        SpecificConfiguration;                  //2
      WORD        LlogicalHeads;                                //3
Застарівший
```

```
            WORD        Retired1[2];                                    //4-5
            WORD        LogicalSectors;                                 //6
Застарівший
            DWORD       ReservedForCompactFlash;                //7-8
            WORD        Retired2;                                       //9
            CHAR        SerialNumber[20];                       //10-19
            WORD        Retired3;                                       //20
            WORD        BufferSize;                                     //21
Застарівший
            WORD        Obsolute4;                                      //22
            CHAR        FirmwareRev[8];                                 //23-26
            CHAR        Model[40];                                      //27-46
            WORD        MaxNumPerInterupt;                              //47
            WORD        Reserved1;                                      //48
            WORD        Capabilities1;                                  //49
            WORD        Capabilities2;                                  //50
            DWORD       Obsolute5;                                      //51-52
            WORD        Field88and7064;                                 //53
            WORD        Obsolute6[5];                                   //54-58
            WORD        MultSectorStuff;                        //59
            DWORD       TotalAddressableSectors;                //60-61
            WORD        Obsolute7;                                      //62
            WORD        MultiWordDma;                                   //63
            WORD        PioMode;                                        //64
            WORD        MinMultiwordDmaCycleTime;               //65
            WORD        RecommendedMultiwordDmaCycleTime;  //66
            WORD        MinPioCycleTimewoFlowCtrl;              //67
            WORD        MinPioCycleTimeWithFlowCtrl;       //68
            WORD        Reserved2[6];                                   //69-74
            WORD        QueueDepth;                                     //75
            WORD        SerialAtaCapabilities;                  //76
            WORD        ReservedForFutureSerialAta;             //77
            WORD        SerialAtaFeaturesSupported;             //78
            WORD        SerialAtaFeaturesEnabled;               //79
            WORD        MajorVersion;                                   //80
            WORD        MinorVersion;                                   //81
            WORD        CommandSetSupported1;                   //82
            WORD        CommandSetSupported2;                   //83
            WORD        CommandSetSupported3;                   //84
            WORD        CommandSetEnabled1;                             //85
            WORD        CommandSetEnabled2;                             //86
            WORD        CommandSetDefault;                              //87
            WORD        UltraDmaMode;                                   //88
            WORD        TimeReqForSecurityErase;                //89
            WORD        TimeReqForEnhancedSecure;               //90
            WORD        CurrentPowerManagement;                 //91
            WORD        MasterPasswordRevision;                 //92
            WORD        HardwareResetResult;                    //93
            WORD        AcoustricManagement;                    //94
            WORD        StreamMinRequestSize;                   //95
            WORD        StreamingTimeDma;                       //96
            WORD        StreamingAccessLatency;                 //97
            DWORD       StreamingPerformance;                   //98-99
            ULONGLONG   MaxUserLba;                                     //100-103
            WORD        StremingTimePio;                        //104
            WORD        Reserved3;                                      //105
            WORD        SectorSize;                                     //106
            WORD        InterSeekDelay;                                 //107
            WORD        IeeeOui;                                        //108
            WORD        UniqueId3;                                      //109
            WORD        UniqueId2;                                      //110
            WORD        UniqueId1;                                      //111
            WORD        Reserved4[4];                                   //112-115
            WORD        Reserved5;                                      //116
            DWORD       WordsPerLogicalSector;                  //117-118
            WORD        Reserved6[8];                                   //119-126
            WORD        RemovableMediaStatus;                   //127
            WORD        SecurityStatus;                                 //128
            WORD        VendorSpecific[31];                             //129-159
```

```
            WORD        CfaPowerMode1;                              //160
            WORD        Reserved7[15];                              //161-175
            CHAR        CurrentMediaSerialNo[60];                   //176-205
            WORD        SctCommandTransport;                        //206  254
            WORD        ReservedForCeAta1[2];                       //207-208
            WORD        AlignmentOfLogicalBlocks;                   //209
            DWORD       WriteReadVerifySectorCountMode3;    //210-211
            DWORD       WriteReadVerifySectorCountMode2;    //212-213
            WORD        NvCacheCapabilities;                        //214
            DWORD       NvCacheSizeLogicalBlocks;                   //215-216
            WORD        NominalMediaRotationRate;                   //217
            WORD        Reserved8;                                  //218
            WORD        NvCacheOptions1;                            //219
            WORD        NvCacheOptions2;                            //220
            WORD        Reserved9;                                  //221
            WORD        TransportMajorVersionNumber;        //222
            WORD        TransportMinorVersionNumber;        //223
            WORD        ReservedForCeAta2[10];                      //224-233
            WORD        MinimumBlocksPerDownloadMicrocode;  //234
            WORD        MaximumBlocksPerDownloadMicrocode;  //235
            WORD        Reserved10[19];                             //236-254
            WORD        IntegrityWord;                              //255
      };
#pragma     pack(pop)

public:
      DWORD UpdateSmartInfo(DWORD index);
      BOOL UpdateIdInfo(DWORD index);
      BYTE GetAamValue(DWORD index);
      BYTE GetApmValue(DWORD index);
      BOOL EnableAam(DWORD index, BYTE param);
      BOOL EnableApm(DWORD index, BYTE param);
      BOOL DisableAam(DWORD index);
      BOOL DisableApm(DWORD index);
      BYTE GetRecommendAamValue(DWORD index);
      BYTE GetRecommendApmValue(DWORD index);

      BOOL Init(BOOL useWmi, BOOL advancedDiskSearch, PBOOL flagChangeDisk);
      BOOL MeasuredTimeUnit();
      DWORD GetPowerOnHours(DWORD rawValue, DWORD timeUnitType);
      DWORD GetPowerOnHoursEx(DWORD index, DWORD timeUnitType);

      struct DISK_POSITION
      {
            INT                         PhysicalDriveId;
            INT                         ScsiPort;
            INT                         ScsiTargetId;
      };

      struct ATA_SMART_INFO
      {
            IDENTIFY_DEVICE         IdentifyDevice;
            SMART_ATTRIBUTE         Attribute[MAX_ATTRIBUTE];
            SMART_THRESHOLD         Threshold[MAX_ATTRIBUTE];

            BOOL                    IsSmartEnabled;
            BOOL                    IsCheckSumError;
            BOOL                    IsWord88;
            BOOL                    IsWord64_76;

            BOOL                    IsSmartSupported;
            BOOL                    IsLba48Supported;
            BOOL                    IsAamSupported;
            BOOL                    IsApmSupported;
            BOOL                    IsAamEnabled;
            BOOL                    IsApmEnabled;
            BOOL                    IsNcqSupported;
            BOOL                    IsNvCacheSupported;
            BOOL                    IsMaxtorMinute;
```

```
		INT			PhysicalDriveId;
		INT			ScsiPort;
		INT			ScsiTargetId;
//		INT			AccessType;

		DWORD			TotalDiskSize;
		DWORD			Cylinder;
		DWORD			Head;
		DWORD			Sector;
		DWORD			Sector28;
		ULONGLONG		Sector48;
		DWORD			DiskSizeChs;
		DWORD			DiskSizeLba28;
		DWORD			DiskSizeLba48;
		DWORD			BufferSize;
		ULONGLONG		NvCacheSize;
		DWORD			TransferModeType;
		DWORD			DetectedTimeUnitType;
		DWORD			MeasuredTimeUnitType;
		DWORD			AttributeCount;
		INT			DetectedPowerOnHours;
		INT			MeasuredPowerOnHours;
		INT			PowerOnRawValue;
		INT			PowerOnStartRawValue;
		DWORD			PowerOnCount;
		DWORD			Temperature;
		double			Speed;

		INT			Life;

		DWORD			Major;
		DWORD			Minor;

		DWORD			DiskStatus;
		DWORD			DriveLetterMap;
		//
		DWORD			AlarmTemperature;
		BOOL			AlarmHealthStatus;

		INTERFACE_TYPE		InterfaceType;
		COMMAND_TYPE		CommandType;

		DWORD			VendorId;
		DWORD			ProductId;

		CString			SerialNumber;
		CString			SerialNumberReverse;
		CString			FirmwareRev;
		CString			FirmwareRevReverse;
		CString			Model;
		CString			ModelReverse;
		CString			ModelWmi;
		CString			ModelSerial;
		CString			DriveMap;
		CString			MaxTransferMode;
		CString			CurrentTransferMode;
		CString			MajorVersion;
		CString			MinorVersion;
		CString			Interface;
		CString			Enclosure;
		CString			CommandTypeString;
	};

	struct EXTERNAL_DISK_INFO
	{
		CString Enclosure;
		DWORD VendorId;
		DWORD ProductId;
```

```
	};

	CArray<ATA_SMART_INFO, ATA_SMART_INFO> vars;
	CArray<EXTERNAL_DISK_INFO, EXTERNAL_DISK_INFO> externals;

	CStringArray m_IdeController;
	CStringArray m_ScsiController;
	CStringArray m_UsbController;
	CString m_ControllerMap;

	BOOL IsEnabledWmi;
	DWORD MeasuredGetTickCount;

protected:
	OSVERSIONINFOEX m_Os;
	CString m_SerialNumberA_Z[26];
	BOOL m_FlagAtaPassThrough;

	BOOL GetDiskInfo(INT physicalDriveId, INT scsiPort, INT scsiTargetId,
INTERFACE_TYPE interfaceType, VENDOR_ID vendorId);
	BOOL AddDisk(INT PhysicalDriveId, INT ScsiPort, INT scsiTargetId,
COMMAND_TYPE commandType, IDENTIFY_DEVICE* identify);
	DWORD CheckSmartAttributeUpdate(DWORD index, SMART_ATTRIBUTE* pre,
SMART_ATTRIBUTE* cur);

	VOID InitAtaInfo();
	VOID InitAtaInfoByWmi();
	VOID InitStruct();
	VOID ChangeByteOrder(PCHAR str, DWORD length);
	BOOL CheckAsciiStringError(PCHAR str, DWORD length);
	HANDLE GetIoCtrlHandle(BYTE index);
	BOOL SendAtaCommand(DWORD i, BYTE main, BYTE sub, BYTE param);

	BOOL DoIdentifyDevicePd(INT physicalDriveId, IDENTIFY_DEVICE* identify);
	BOOL GetSmartAttributePd(INT physicalDriveId, ATA_SMART_INFO* asi);
	BOOL GetSmartThresholdPd(INT physicalDriveId, ATA_SMART_INFO* asi);
	BOOL ControlSmartStatusPd(INT physicalDriveId, BYTE command);
	BOOL SendAtaCommandPd(INT physicalDriveId, BYTE main, BYTE sub, BYTE
param, PBYTE data, DWORD dataSize);

	BOOL DoIdentifyDeviceScsi(INT scsiPort, INT scsiTargetId, IDENTIFY_DEVICE*
identify);
	BOOL GetSmartAttributeScsi(INT scsiPort, INT scsiTargetId, ATA_SMART_INFO*
asi);
	BOOL GetSmartThresholdScsi(INT scsiPort, INT scsiTargetId, ATA_SMART_INFO*
asi);
	BOOL ControlSmartStatusScsi(INT scsiPort, INT scsiTargetId, BYTE command);
	BOOL SendAtaCommandScsi(INT scsiPort, INT scsiTargetId, BYTE main, BYTE
sub, BYTE param);

	BOOL DoIdentifyDeviceSat(INT physicalDriveId, IDENTIFY_DEVICE* identify,
COMMAND_TYPE commandType);
	BOOL GetSmartAttributeSat(INT physicalDriveId, ATA_SMART_INFO* asi);
	BOOL GetSmartThresholdSat(INT physicalDriveId, ATA_SMART_INFO* asi);
	BOOL ControlSmartStatusSat(INT physicalDriveId, BYTE command, COMMAND_TYPE
commandType);
	BOOL SendAtaCommandSat(INT physicalDriveId, BYTE main, BYTE sub, BYTE
param, COMMAND_TYPE commandType);

	DWORD CheckDiskStatus(SMART_ATTRIBUTE* attribute, SMART_THRESHOLD*
threshold, DWORD attributeCount, DWORD vendorId);

	DWORD GetTransferMode(WORD w63, WORD w76, WORD w88, CString
&currentTransferMode, CString &maxTransferMode, CString &Interface,
INTERFACE_TYPE *interfaceType);
	DWORD GetTimeUnitType(CString model, CString firmware, DWORD major, DWORD
transferMode);
	DWORD GetAtaMajorVersion(WORD w80, CString &majorVersion);
//	DWORD GetMaxtorPowerOnHours(DWORD currentValue, DWORD rawValue);
```

```
        static int Compare(const void *p1, const void *p2);
};
```

Файл AboutDlg.cpp – довідка

```
#include "stdafx.h"
#include "DiskInfo.h"
#include "AboutDlg.h"

IMPLEMENT_DYNCREATE(CAboutDlg, CDHtmlDialog)

CAboutDlg::CAboutDlg(CWnd* pParent /*=NULL*/)
                  : CDHtmlDialogEx(CAboutDlg::IDD, CAboutDlg::IDH, pParent)
{

}

CAboutDlg::~CAboutDlg()
{
}

BOOL CAboutDlg::OnInitDialog()
{
                  CDHtmlDialogEx::OnInitDialog();

                  InitDHtmlDialog(SIZE_X, SIZE_Y, ((CDiskInfoApp*)AfxGetApp())-
>m_AboutDlgPath);

                  return TRUE;
}

void CAboutDlg::OnDocumentComplete(LPDISPATCH pDisp, LPCTSTR szUrl)
{
                  CString cstr;
                  cstr = szUrl;
                  if(cstr.Find(_T("html")) != -1 || cstr.Find(_T("dlg")) != -1)
                  {
                        m_FlagShowWindow = TRUE;
                       m_Copyright = PRODUCT_COPYRIGHT;
                        UpdateData(FALSE);
                        ShowWindow(SW_SHOW);
                  }
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDHtmlDialogEx)
END_MESSAGE_MAP()

BEGIN_DHTML_EVENT_MAP(CAboutDlg)
                  DHTML_EVENT_ONCLICK(_T("CrystalDewWorld"), OnCrystalDewWorld)
END_DHTML_EVENT_MAP()

HRESULT CAboutDlg::OnCrystalDewWorld(IHTMLElement* /*pElement*/)
{
                  if(GetUserDefaultLCID() == 0x0411)// Japanese
                  {
                        ShellExecute(NULL, NULL, URL_CRYSTAL_DEW_WORLD_JA, NULL,
NULL, SW_SHOWNORMAL);
                  }
                  else // Other Language
                  {
                        ShellExecute(NULL, NULL, URL_CRYSTAL_DEW_WORLD_EN, NULL,
NULL, SW_SHOWNORMAL);
                  }

                  return S_FALSE;
}
```

Файл AboutDlg.h – бібліотека для файлу AboutDlg.cpp

```
#pragma once

class CAboutDlg : public CDHtmlDialogEx
{
	DECLARE_DYNCREATE(CAboutDlg)

	static const int SIZE_X = 240;
	static const int SIZE_Y = 200;

public:
	CAboutDlg(CWnd* pParent = NULL);
	virtual ~CAboutDlg();

	enum { IDD = IDD_ABOUT, IDH = IDR_HTML_DUMMY };

protected:
	virtual void DoDataExchange(CDataExchange* pDX);
	virtual BOOL OnInitDialog();
	virtual void OnDocumentComplete(LPDISPATCH pDisp, LPCTSTR szUrl);

	CString m_Version;
	CString m_Release;
	CString m_Copyright;

	HRESULT OnCrystalDewWorld(IHTMLElement *pElement);

	DECLARE_MESSAGE_MAP()
	DECLARE_DHTML_EVENT_MAP()
};
```