ViaThinkSoft UserDetect2

Daniel Marschall, 3 January 2017

Table of Contents

What is UserDetect2?

UserDetect2 is a program that allows the user to execute different programs depending on their current environment (e.g. MAC addresses, user name or computer name), so that a single executable file, e.g. shared over a network drive, a flash drive or external hard disk, can perform tasks for different work stations. The environment identifications can be extended by plugins.

Usage example

You have an external hard disk which you use for a daily backup with a backup tool (e.g. Microsoft RoboCopy).

You use this external drive to perform backups for different computers.

Additionally, you want to decide if the computer should be shutdown after the backup, or not. (Can be useful if you leave the computer alone, while the backup is performing)

If you have 2 computers with the names “JohnPC” and “JohnLaptop”, then you would probably need 4 batch files:

E:\JohnPC\backup_no_shutdown.bat

E:\JohnPC\backup_shutdown.bat

E:\JohnLaptop\backup_no_shutdown.bat

E:\JohnLaptop\backup_shutdown.bat

If you accidently start the wrong batch file, the backups will be inconsistent, and there may be data loss.

But if you use UserDetect2, you could create following Task Definition File:

```
[NoShutdown]
Description=Run backup without shutdown
ComputerName:JohnPC=JohnPC\backup_no_shutdown.bat
ComputerName:JohnLaptop=JohnLaptop\backup_no_shutdown.bat

[Shutdown]
Description=Run backup and shutdown
ComputerName:JohnPC=JohnPC\backup_shutdown.bat
ComputerName:JohnLaptop=JohnLaptop\backup_shutdown.bat
```

In this case, you would only need to run “E:\UserDetect2.exe” (maybe even use it as AutoRun application, if you are working with Windows Vista or previous versions of Windows) and then select if you want to perform a backup with or without shutdown. UserDetect2 will select the correct batch file for you.

Command Line usage

The syntax can be retrieved by calling

```
UserDetect2.exe /?
```

Syntax:

```
UserDetect2.exe
[TaskDefinitionFile [/T TaskName] | /C IdentificationTerm [Command] | /?]
```

Examples:

`UserDetect2.exe` without parameters

> The GUI of UserDetect2 will be used.

```
UserDetect2.exe Foo.ini
```

> The GUI of UserDetect2 will be used, and the Task Definition File Foo.ini will be used (the default Task Definition File.

```
UserDetect2.exe Foo.ini /T ExampleTask1
```

> The GUI will not be used. Instead, the task ExampleTask1 of the Task Definition File Foo.ini will be called immediately.

```
UserDetect2.exe /C ComputerName:JohnPC
```

> The GUI will not be used. A single check will be performed to check if the plugin providing the identification "ComputerName" returns "JohnPC". In this case, the return code will be 0. Otherwise, it will be 1. You can query these return codes with "IF ERRORLEVEL" in Windows's batch files.

```
UserDetect2.exe /C ComputerName:JohnPC $ADMIN$calc.exe
```

> If the computer's name is JohnPC, the program calc.exe will be launched with administrator privileges. The return codes 0 and 1 are still used like described above. Additionally, following return codes are possible due to the launch of the application:
>
> `EXITCODE_RUN_FAILURE (2)`
>
> `EXITCODE_SYNTAX_ERROR (13)`

Return codes (ErrorLevel)

`EXITCODE_OK = 0`

> No error was reported.

`EXITCODE_TASK_NOTHING_MATCHES = 1`

> The task was found, but no definition matched the current environment.

```
EXITCODE_RUN_FAILURE = 2
```

The task was found, but at least one executable was not found resp. couldn't be launched.

```
EXITCODE_TASK_NOT_EXISTS = 10
```

The specified task does not exist.

```
EXITCODE_INI_NOT_FOUND = 11
```

The specified Task Definition file does not exist.

```
EXITCODE_RUNCMD_SYNTAX_ERROR = 12
```

The syntax of the command defined in the Task Definition File is wrong.

In special, the opening quote sign does not have a matching a closing quote sign.

```
EXITCODE_SYNTAX_ERROR = 13
```

The syntax of the arguments passed to UserDetect2.exe is wrong.

Task Definition File (UserDetect2.ini)

The default **Task Definition File** is called UserDetect2.ini . A different INI file can be used using a Command Line argument. If no file with the name UserDetect2.ini or <ApplicationName>.ini file is existing, and no command line argument is used, the GUI will ask the user for the path of the Task Definition File.

Here is an example of a Task Definition File:

```
[ExampleTask1]
Description=Run Task #1
Icon=%SystemRoot%\system32\Shell32.dll,3
CloseAfterLaunching=true
ComputerName:JohnPC=calc.exe
LAN_MAC:66-55-44-33-22-11=notepad.exe
AccountSID:S-1-5-21-3669290038-3293053376-393244323-1000=calc.exe

[ExampleTask2]
Description=Run Task #2
ComputerName:JohnLaptop=calc.exe
LAN_MAC:11-22-33-44-55-66=notepad.exe
AccountSID:S-1-5-21-3669290038-3293053376-393244323-1001=calc.exe
```

Comments

The Task Definition File allows comments. These comment lines must have a ";" at the beginning of the line, for example:

```
[MyTask1]
; Description will be shown in the GUI
Description=Example
```

Section fields

Each section starts with a name in square brackets, e.g. `[ExampleTask1]`. There can be multiple sections. Each section defines a task.

Field `Description`

The description is an optional field, but is highly recommended. In the GUI, it will be shown as name for the task.

Field `Icon`

This is an optional field, to replace the icon of the task with a different icon. The syntax is:

`Icon: <IconDLL>,<IconIndex>`

For example, `Icon=%SystemRoot%\system32\Shell32.dll,3`

Will use Icon #3 of the file Shell32.dll (environment variables like `%SystemRoot%` are resolved).

Field `CloseAfterLaunching`

This is an optional field. If the value is `true` (or `yes`, or `1`), then the GUI will close once the task is started. Otherwise, you can write `false` (or `no`, or `0`), or remove the field. The default value is `false`.

Field `WarnIfNothingMatches.GUI`

This is an optional field. If the value is `true` (or `yes`, or `1`), then a warning dialog will be shown if a task is called from the GUI, but no application could be started, because no identification matches the current environment, or if no identification fields are added in the current task. Otherwise, you can write `false` (or `no`, or `0`), or remove the field. The default value is `true`.

Field `WarnIfNothingMatches.CLI`

Same as `WarnIfNothingMatches.GUI`, but this field applies to the command line invocation (parameter `/T`). The default value is `false`.

Identification fields

Identification fields are located in the Task section, too. Each identification field has following syntax:

`<IdentificationMethodName>:<IdentificationString>=<Command> <Optional Parameters>`

Example:

> `ComputerName:John=calc.exe`
>
> If the plugin with the identification method name "ComputerName" will returns the identification "John", then the program "calc.exe" will be started.

The short name of the plugins, as well as the identification strings can be seen in the GUI in the "Identifications" tab. The GUI can also create a Task Definition File (INI file) template for you.

Alternatively to the `MethodName`, you can use the GUID of the plugin. The syntax is `{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}`.

Attention:

1. **`IdentificationMethodName` and `IdentificationString` are case insensitive! To avoid this, you can include the modifier `$CASESENSITIVE$` somewhere inside the identification string. Please note that if you have concatenated multiple identification strings (via &&), `$CASESENSITIVE$` must be added to every sub-term.**
2. **Although it is recommended to name the DLL file equally to the identification method name, there is no guarantee that they match. Only the identification method name will be used, not the name of the DLL file!**

Notes about the command launching:

- If multiple identifications are matching (even if the program is the same), the programs will be called anyway.
- Environment variables like `%SystemRoot%` are resolved.
- Relative paths are allowed.
- A path with white spaces (e.g. "C:\Program Files\xyz.exe") must be enclosed in quotes.
- Also non-executable files, like Word-Documents, can be executed.

"Run as Administrator" flag

In case you want to run an application as administrator, you can use following syntax.

`LAN_MAC:11-22-33-44-55-66=`**`$ADMIN$`**`test\example.bat`

If your application does not run with administrator privileges, then you will be asked using the UAC dialog of Windows.

You can combine $ADMIN$ and $RIOD$.

"Run in own directory" compatibility prefix

Some applications or batch files require that they are called in their own directory (this is technically a bug), and will fail if the working directory is different. In this case, you can use following special syntax:

`LAN_MAC:11-22-33-44-55-66=`**`$RIOD$`**`test\example.bat`

The prefix `$RIOD$` stands for "Run-in-own-directory". In this case, the working directory of example.bat will be test\, instead of the directory where UserDetect2.exe was started from.

Note:

If your application or batch file is intolerant to different working directories, please consider following to solve this bug – it would be in your interest:

In Batch files (*.bat), use `%~dp0` to receive the directory name where the script is located in (including trailing path delimiter).

In Delphi projects, use `IncludeTrailingPathDelimiter(ExtractFilePath(ParamStr(0)))` to receive this path. You may want to pass this path into `SetCurrentDir()`.

You can combine $ADMIN$ and $RIOD$

Concatenations

Multiple identification criteria can be merged with an "AND" condition. This is done with the expression "&&".

Example:

```
ComputerName:ComputerB&&UserName:John=calc.exe
```

Calc.exe will be called at ComputerB, but only when User John is logged in.

Troubleshooting

Error message "The plugin "Plugins\<Plugin1>.dll" and the plugin "Plugins\<Plugin2>.dll" have the same identification GUID. The latter will not be loaded.

This error message can have 2 reasons:

Reason 1

You have the same plugin twice, with different file names. In this case, you should delete on of these files.

Solution: You might want to keep the file with the highest version number and/or modification date.

Reason 2

The plugin developer accidently used a GUID twice, or didn't change the GUID while using a code template. In this case you have to create a workaround.

Solution: Create a new INI file with the name Plugins\<Name>.ini where <Name> is the filename of one of the conflicting plugins. For example, if the DLL file is named Plugins\Example.dll , the INI file would be named Plugins\Example.ini . This INI file should have following contents:

```
[Compatibility]
OverrideGUID={936DA01F-9ABD-4D9D-80C7-02AF85C822A8}
```

Where `<GUID>` is a GUID in the format `{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}`.

You can generate a new GUID at this website:
https://misc.daniel-marschall.de/tools/uuid_mac_decoder/

You have to copy the field "Your UUID" and add curly braces around it.

Note: `AutoOSNotSupported` and `OverrideGUID` can be combined.

Please also contact the plugin authors and inform him about the issue.

Error message "Entry could not be located" or "One of the library files needed to run this application cannot be found."

If a plugin fails loading with these error messages, it is most likely that your Operation System does not provide a DLL or its functions which is required for the plugin. Usually, the developers of the plugin should detect such incompatibilities and return appropriate status codes. However, very old Windows versions are likely to be not checked by plugin authors.

To disable this error message, you have 2 options:

1. Delete the plugin file. This might be a problem if you want to use the plugin at different computers which have the missing DLL installed.
2. Add a manifest file to manually disable the error message. The status "OS not supported" will be set, if one of these two errors appear.

   Create a new INI file with the name Plugins\<Name>.ini where <Name> is the filename of one of the conflicting plugins. For example, if the DLL file is named Plugins\Example.dll, the INI file would be named Plugins\Example.ini. This INI file should have following contents:

   ```
   [Compatibility]
   AutoOSNotSupported=1
   ```

   Note: `AutoOSNotSupported` and `OverrideGUID` can be combined.

   Note: The value `AutoOSNotSupported=2` will extend the overriding of the status for all errors of loading a plugin DLL. The value `AutoOSNotSupported=3` will additionally convert all Failure status codes into the NotAvailable "OS not supported" status.

   Please also contact the plugin authors and inform him about the issue.

Plugin-Development

Plugins are automatically loaded. Please name your DLL file identically to the identification method name you will use (see `IdentificationMethodNameW`).

Location of plugin files

UserDetect2 will detect following files as plugins:

- *.udp
- Plugins*.udp
- Plugins*.dll

SDK Contents

UserDetect2 as well as the default plugins are developed in Delphi. However, plugins can be written in any programming language like C++. The SDK provides header files for Delphi and C.

Delphi units:

- UD2_PluginUtils.pas
- UD2_PluginIntf.pas
- UD2_PluginStatus.pas

C header files:

- ud2_api.h
- ud2_utils.h (included via ud2_api.h)
- ud2_guid.h (included via ud2_api.h)
- ud2_status.h (included via ud2_api.h)

Exported functions

A plugin has to export following methods, whose names are NOT decorated.

In Delphi notation:

```
function PluginInterfaceID(): TGUID; cdecl;

function PluginIdentifier(): TGUID; cdecl;

function PluginNameW(lpPluginName: LPWSTR; cchSize: DWORD; wLangID:
LANGID): UD2_STATUS; cdecl;

function PluginVersionW(lpPluginVersion: LPWSTR; cchSize: DWORD;
wLangID: LANGID): UD2_STATUS; cdecl;

function PluginVendorW(lpPluginVendor: LPWSTR; cchSize: DWORD; wLangID:
LANGID): UD2_STATUS; cdecl;

function CheckLicense(lpReserved: LPVOID): UD2_STATUS; cdecl;

function IdentificationMethodNameW(lpIdentificationMethodName: LPWSTR;
cchSize: DWORD): UD2_STATUS; cdecl;

function IdentificationStringW(lpIdentifier: LPWSTR; cchSize: DWORD):
UD2_STATUS; cdecl;

function DescribeOwnStatusCodeW(lpErrorDescription: LPWSTR; cchSize:
DWORD; statusCode: UD2_STATUS; wLangID: LANGID): BOOL; cdecl;
```

In C notation:

```
__cdecl GUID PluginInterfaceID();
```

```
__cdecl GUID PluginIdentifier();

__cdecl UD2_STATUS PluginNameW(LPWSTR lpPluginName, DWORD cchSize,
LANGID wLangID);

__cdecl UD2_STATUS PluginVersionW(LPWSTR lpPluginVersion, DWORD
cchSize, LANGID wLangID);

__cdecl UD2_STATUS PluginVendorW(LPWSTR lpPluginVendor, DWORD cchSize,
LANGID wLangID);

__cdecl UD2_STATUS CheckLicense(LPVOID lpReserved);

__cdecl UD2_STATUS IdentificationMethodNameW(LPWSTR
lpIdentificationMethodName, DWORD cchSize);

__cdecl UD2_STATUS IdentificationStringW(LPWSTR lpIdentifier, DWORD
cchSize);

__cdecl BOOL DescribeOwnStatusCodeW(LPWSTR lpErrorDescription, DWORD
cchSize, UD2_STATUS statusCode, LANGID wLangID);
```

Method `PluginInterfaceID`

This method has to return the GUID `{6C26245E-F79A-416C-8C73-BEA3EC18BB6E}` which is stored in the constant `USERDETECT2_IDPLUGIN_V1`.

Method `PluginIdentifier`

This method has to return a GUID which is unique for this plugin.

Method `IdentificationStringW`

The function returns the identification string, which is a null-terminated C-string which contains the value of the identification performed. For example, the UserName-Plugin would output "`John<NUL>`". With this output, the task "`UserName:John=calc.exe`" would be performed (calc.exe will be called).

Annotations:

1. Empty string = no identification could be performed, e.g. the MAC-Plugin cannot work on a computer without network interfaces. In this case, the output of the plugin will be IGNORED. For example, for the MAC-plugin, the task "`LAN_MAC:=calc.exe`" will NOT be executed, because the identification string (after the colon) cannot be empty.
2. Multiple outputs: Multiple identifications, e.g. multiple MAC addresses are separated with a linefeed (0x10). Each identification will be traded separately. For example, if the MAC-Plugin returns "`11-22-33-44-55-66<LF>66-55-44-33-22-11<NUL>`", the following tasks will be executed:

   ```
   LAN_MAC: 11-22-33-44-55-66=calc.exe

   LAN_MAC:66-55-44-33-22-11=notepad.exe
   ```

The identification string must be a wide string (16 bit).

The string must not contain an equal-sign (=) or "&&".

The output will be treated case sensitive.

Methods `PluginNameW, PluginVersionW, PluginVendorW`

These methods return a human readable name, vendor or version of the plugin. These strings will be shown in the GUI.

`wLangID` is the 16-bit language ID according to the WinAPI:

Lower 8 bit = Primary Language ID (e.g. `LANG_ENGLISH`)

Upper 8 bit = Sublanguage ID (e.g. `SUBLANG_ENGLISH_US`)

The plugin name must be a wide string (16 bit).

Method `CheckLicense`

This method will be called before the plugin is loaded. The plugin has the ability to check if it may be used or not. For example, a plugin with a time-limited license can check the computer clock, or a plugin for Beta-Testers can check if it is running on the intended computer only.

The return value should be either `UD2_STATUS_OK_LICENSED` or `UD2_STATUS_FAILURE_PLUGIN_NOT_LICENSED`, but other error codes can be returned, too.

Method `IdentificationMethodNameW`

This method returns a short string of the identification method. This name will be used in the identifications lines in the Task Definition File. For example, the MAC-Addresses plugin will have the method name "`LAN_MAC`", so that a task "`LAN_MAC:11-22-33-44-55-66=calc.exe`" can be defined in the Task Definition File.

The identification method name must be a wide string (16 bit).

The string must not contain an equal-sign (=) or "&&".

The output will be treated case sensitive.

Method `DescribeOwnStatusCodeW`

This function should return human readable descriptions of the non-generic status codes which THIS plugin is returning. You do not need to describe generic status codes (because they are already described of the UserDetect2 API). This function will only be invoked, once a status code could not be resolved. Return TRUE, if you could describe the code, FALSE otherwise, e.g. if the status code is unknown or if the buffer is too small.

Attention: If the requested language is not available, please return TRUE, and return the description in English.

Status codes

Structure of status codes (`UD2_STATUS`)

Delphi (UD2_PluginStatus.pas)

```
type
  UD2_STATUSCAT       = WORD;
  UD2_STATUSAUTH      = TGUID;
  UD2_STATUSMSG       = DWORD;
  UD2_STATUSEXTRAINFO = DWORD;

  UD2_STATUS = packed record
    cbSize:      BYTE;
    bReserved:   BYTE;
    wCategory:   UD2_STATUSCAT;
    grAuthority: UD2_STATUSAUTH;
    dwMessage:   UD2_STATUSMSG;
    dwExtraInfo: UD2_STATUSEXTRAINFO;
  end;
```

C++ (ud2_status.h)

```
typedef WORD  UD2_STATUSCAT;
typedef GUID  UD2_STATUSAUTH;
typedef DWORD UD2_STATUSMSG;
typedef DWORD UD2_STATUSEXTRAINFO;

#pragma pack(push, 1) // no alignment
typedef struct _UD2_STATUS {
    BYTE                cbSize;
    BYTE                bReserved;
    UD2_STATUSCAT       wCategory;
    UD2_STATUSAUTH      grAuthority;
    UD2_STATUSMSG       dwMessage;
    UD2_STATUSEXTRAINFO dwExtraInfo;
} UD2_STATUS;
#pragma pack(pop) // restore previous pack value
```

`cbSize` Size of struct

The size of the struct, which is currently 16, but you should use `sizeof(UD2_STATUS)`.

`bReserved` Reserved byte

Should be always zero.

`wCategory` Category

`0x0000` = Success
`0x0001` = Not available
`0x0002` = Error
`0x0003..0xFFFF` are reserved

grAuthority Authority

{90F53368-1EFB-4350-A6BC-725C69938B9C} is the GUID for generic status codes. You should use them, if applicable. If you want to return individual status codes, please generate a new GUID.

dwMessage Message

0x00000000 = an unspecified status in category "C", issued by authority "A"
0x00000001..0xFFFFFFFF = the status in category "C", issued by authority "A".

dwExtraInfo Extra Information

This field can be used to pass additional information, for example, a pointer to shared memory, or passing an error code of a failed external API (e.g. WinAPI).

Currently assigned generic status codes

Category "Successful"

UD2_STATUS_OK_UNSPECIFIED = Unspecified success code.
UD2_STATUS_OK_SINGLELINE = Success, one identifier returned.
UD2_STATUS_OK_MULTILINE = Success, multiple identifiers returned.
UD2_STATUS_OK_LICENSED = The plugin is licensed.

Category "Not available"

UD2_STATUS_NOTAVAIL_UNSPECIFIED = Not available, Unspecified.
UD2_STATUS_NOTAVAIL_OS_NOT_SUPPORTED = Operating system not supported.
UD2_STATUS_NOTAVAIL_HW_NOT_SUPPORTED = Hardware not supported.
UD2_STATUS_NOTAVAIL_NO_ENTITIES = No entities available, e.g. no network interfaces for identifying MAC addresses.
UD2_STATUS_NOTAVAIL_WINAPI_CALL_FAILURE = An API call has failed, possibly operating system not supported. dwExtraInfo contains the Windows Status Code (e.g. from GetLastError).
STATUS_NOTAVAIL_ONLY_ACCEPT_DYNAMIC = The plugin does only accept dynamic requests.
STATUS_NOTAVAIL_INVALID_INPUT = The plugin received an invalid input (i.e. dynamic data).
UD2_STATUS_NOTAVAIL_DOES_NOT_ACCEPT_DYNAMIC_REQUESTS = The plugin does not allow dynamic requests

Category "Failure"

UD2_STATUS_FAILURE_UNSPECIFIED = Unspecified error.
UD2_STATUS_FAILURE_BUFFER_TOO_SMALL = Buffer too small.
UD2_STATUS_FAILURE_INVALID_ARGS = Invalid arguments passed to an internal function.
UD2_STATUS_FAILURE_PLUGIN_NOT_LICENSED = The plugin is not licensed.
UD2_STATUS_FAILURE_NO_RETURNED_VALUE = Plugin did not return a status
UD2_STATUS_FAILURE_CATCHED_EXCEPTION = Catched unexpected Exception

Additional functions in Delphi (PluginStatus.pas)

```
function UD2_STATUS_FormatStatusCode(grStatus: UD2_STATUS): string;

function UD2_STATUS_Equal(grStatus1, grStatus2: UD2_STATUS;
compareExtraInfo: boolean): boolean;

function UD2_STATUS_OSError(OSError: DWORD): UD2_STATUS;
```

Relevant code part in C notation (ud2_status.h, included via ud2_api.h)

```
int UD2_STATUS_FormatStatusCode(char* szStr, size_t cchLen, UD2_STATUS
grStatus);

bool UD2_STATUS_Equal(UD2_STATUS grStatus1, UD2_STATUS grStatus2, bool
compareExtraInfo);

bool operator==(const UD2_STATUS& lhs, const UD2_STATUS& rhs);

UD2_STATUS UD2_STATUS_OSError(DWORD dwOSError);
```

Helper methods

The plugin SDK contains methods which help you writing the strings in the buffer, as well as checking for the buffer length. These functions also return the correct status codes.

In C (ud2_utils.h, included via ud2_api.h):

```
UD2_STATUS UD2_WriteStrW(LPWSTR lpDest, DWORD cchDestSize, LPCWSTR
lpSrc);

#define __GUID(x) GUIDFromLPCWSTR(L ## x)
GUID GUIDFromLPCWSTR(LPCWSTR lpcstrGUID);
```

In Delphi (UD2_PluginUtils.pas):

```
function UD2_WritePascalStringToPointerW(lpDestination: LPWSTR;
     cchSize: DWORD; stSource: WideString): UD2_STATUS;

function UD2_WriteStringListToPointerW(lpDestination: LPWSTR;
     cchSize: DWORD; slSource: TStrings): UD2_STATUS;
```

Explanation:

- The return value of these functions is an `UD2_STATUS` which is either
    - `UD2_STATUS_FAILURE_BUFFER_TOO_SMALL`
    - `UD2_STATUS_NOTAVAIL_UNSPECIFIED` (when no identifier is provided)
    - `UD2_STATUS_OK_MULTILINE`
    - `UD2_STATUS_OK_SINGLELINE`
- `UD2_WriteStrW()` and `UD2_WritePascalStringToPointerW()` provide the same functionality.

- `UD2_WriteStringListToPointerW()` is used for `IdentificationStringW()`, to return multiple identifications. The function will split each `TStrings` entry with `UD2_MULTIPLE_ITEMS_DELIMITER` (0x10). A pendant for C is currently not implemented.

- `__GUID()` will convert a string literal to a `GUID`. This improves the readability of the source code, since the GUID does not have to be written in a `const struct` expression.
  In Delphi, this helper method is not necessary, since the compiler is able to convert a GUID-string to a GUID structure (record) at compile time.

Migration from testuser (predecessor)

UserDetect 2.1 has all identification methods that existed in **ViaThinkSoft testuser**.

ViaThinkSoft testuser	**ViaThinkSoft UserDetect 2.1**
Testuser.exe ":USER:" "xxx"	UserDetect2.exe /C UserName:xxx
Testuser.exe ":COMP:" "xxx"	UserDetect2.exe /C ComputerName:xxx
Testuser.exe ":SID:" "xxx"	UserDetect2.exe /C AccountSID:xxx
Testuser.exe ":HOME:" "xxx"	UserDetect2.exe /C HomeDir:xxx
Testuser.exe ":HOMESHARE:" "xxx"	UserDetect2.exe /C HomeShare:xxx
Testuser.exe ":HOMECOMP:" "xxx"	UserDetect2.exe /C HomeDirUNC:xxx

Known Issues

Parsing problems

- Due to a problem with Delphi's "ReadSectionValues", the comparison of identification strings will not address/compare whitespaces at the beginning or end. So, for example, following INI entries have the same effect:

```
DriveSerial(c:):2SHSWNHA010807 X    =calc.exe
DriveSerial(c:):2SHSWNHA010807 X=calc.exe
```

Compatibility issues with older Windows versions

Currently, the EXE and all plugin DLLs are compiled with Delphi 10.1 Berlin, and are therefore Unicode compatible. Alas, they are not compatible with older Windows versions anymore.

In future, a Non-Unicode EXE file should be published beside the Unicode release. This Non-Unicode release would be compiled with an old Delphi version (e.g. Turbo Delphi 2006), so that the compatibility for older Windows versions (back to Windows 95) can be achieved again.

Compatibility issues with Windows 95

- Icons of the jobs are not shown.

- Error message "One of the library files needed to run this application cannot be found" for the following plugins (IpHlpApi.dll: GetAdaptersInfo is not available)

  - DHCP_IP.dll

  - DHCP_MAC.dll

- LAN_IP.dll
- LAN_MAC.dll
- GatewayIP.dll
- GatewayMAC.dll

- No compatibility with Windows 95 if compiled with Delphi 7. (Problem with image list binary format)

Compatibility issues with Windows NT4 SP6

- The plugins DHCP_IP, DHCP_MAC, GatewayIP, GatewayMAC, LAN_IP are not working because the WinAPI function `GetAdaptersInfo()` is not supported (`ERROR_NOT_SUPPORTED`).

Changelog

2.3.2	3 Jan 2017	Icons are now grayed out when the task is not runnable at the system.
2.3.1	16 Oct 2016	Small bugfixes
2.3	13 Oct 2016	Unicode ready (Temporarily) dropped support for old Delphi and Windows versions
2.2	24 Jul 2016	Introduced dynamic Plugins! WarnIfNothingMatches is now split into: WarnIfNothingMatches.GUI (default true) and WarnIfNothingMatches.CLI (default false). New plugins: TestDynamicEcho (pure dynamic) DriveSerial (pure dynamic)
2.1	18 Jun 2016	Added `$ADMIN$` prefix (at command). Added `$CASESENSITIVE$` prefix (for identification string). Allow loading of *.udp files as alternative to *.dll files. Extended command line usage (/T and /C modes) In command line mode, the main window doesn't flicker. New plugins: WinBuildNumber WinMajorVersion WinMinorVersion WinVersion UserName

		HomeDir HomeDirUNC HomeShare MachineSID EnvironmentString
		Fixed update download.
		Improved handling of erroneous plugin DLLs.
		Added compatibility setting `AutoOSNotSupported`.
		Smaller bugfixes.
2.0.1	28 Feb 2016	Compatibility bugfix for Windows Server 2008
2.0	03 Oct 2015	Initial release

Contact

UserDetect2 is a project of ViaThinkSoft (www.viathinksoft.com) and is released under the terms of the Apache 2.0 license.

Author:

Daniel Marschall

eMail: info@daniel-marschall.de

Web: www.daniel-marschall.de