

# JEDI Windows Security Code Library

## some points about the implementation

Please read the lines below to get an understanding how to write your code. It helps you to improve the source code, the readability and the documentation. Try to stick as much as possible and ask your tutor or JEDI Administrators if there are problems, additions and more.

- First read the file : "*JEDI StyleGuide.pdf*" from the Subversion folder *jwscl/trunk/documentation* on your harddisk.
- Try to stick to the styleguid as best as possible.
- Don't use a source formatter that uses a format that differs too much from the existing source format
- JWSCL can be compiled with **Delphi 7** and newer. Do not use new Delphi constructs without using compiler IFDEFS defined in *jedi.inc* (automatically included by *jwscl.inc*) like **DELPHI11\_UP**.  

```
{IFDEF DELPHI11_UP}  
  //new Delphi code  
{ELSE}  
  //backwards compatible code  
{ENDIF DELPHI11_UP}
```
- If you want to use a new Delphi construct you must make sure that a programmer with an old Delphi version has also the possibility to use the feature in another way.
- There is no FreePascal support at the moment.
- Add your exceptions to *JwsclExceptions.pas*.
- Add your simple types to *JwsclTypes.pas*.
- Add your constants to *JwsclConstants.pas*.
- Add utility functions to *JwsclUtils.pas*.
- Add COM utility to *JwsclComUtils.pas*.
- Name your classes with Jw  
TJwYourClass = class
- Use *TJwString* and *TJwPChar* if you support Ansi- and Unicode. Otherwise use *AnsiString*, *WideString* and *PAnsiChar* and *PWideChar* instead of simple *String* and *PChar*.
- If you need to call a WinAPI function that is Ansi- and Unicode aware you must implement them both instead of calling the generic function<sup>1</sup>.  
If {IFDEF UNICODE} CreateProcessW {ELSE} CreateProcessA {ENDIF UNICODE}  
(TJwPChar(Name),...) then ...  
You can use TJwPChar to automatically cast to PWideChar or PAnsiChar depending on the **UNICODE** directive.
- Add string constants to *JwsclResource*.

---

<sup>1</sup> The JEDI Windows API implements automatic creation of Ansi- or Unicode functions. Most times a Windows API function is implemented three times. E.g. CreateProcessA (AnsiCode), CreateProcessW (Unicode) and CreateProcess that points to CreateProcessA or CreateProcessW depending on the compiler directive UNICODE. If it is set CreateProcessW is used; otherwise CreateProcessA. However JWSCL does only rely on its on UNICODE directive.

- Use the whole power of the exception constructors:

example for a failed windows api function call

```
raise EJwscIWinCallFailedException.CreateFmtWinCall(
    RsSecurityDescriptorInvalid,      //Message as a resource string
    'GetJobObjectInformationLength',  //Source procedure name
    ClassName,                        //Source class name
    RsUNTSid,                         //Source unit file name
    0,                                //Source line (set to 0)
    True,                             //add GetLastError information?
    'QueryInformationJobObject',      //Windows API function name that failed
    ['...'])                          //format string parameters for message
```

- Instead of returning an error code, always raise an exception.
- Create your own meaningful exception classes derived from EJwscISecurityException or any of its descendants.
- Never use the EJwscISecurityException in a raise statement.
- Do not use WinAPI Calls in a JwscIException call since it may change the GetLastError value before the constructor can read it. Instead save the GetLastError manually in a variable and reset it by calling SetLastError.
- Try to check for specific exceptions that you can handle. Do not catch generic exceptions like "Exception" and "EJwscISecurityException".

#### Header

+adapt the unit header if necessary. Add your name as a (co)-author  
 +add description of the unit  
 +add a description of known bugs

#### Documentation

+document your code if it is problematic to understand  
 +document all constants, functions, procedures, types, classes, protected, public and published methods and properties. It is not necessary to document private parts that are only helper methods or variables for properties (which are already documented).  
 +Use XML and/or JavaDoc documentation style. The help creator Doc-O-Matic is used to grab all documentation and to create a documentation file.

Example (pls use only comment brackets "{" for documentation):

More examples can be found at the Doc-O-Matic site <http://www.doc-o-matic.com/examplesourcecode.html>

Be aware that the JWSCL doc headers may differ from the shown examples because they were translated from another (incompatible) comment style.

**<B>IsProcessInJob</B>** returns whether a process is assigned to the job or not.

@param hProcess defines any handle to the process that is tested for membership.

@param Returns true if the process is a member of the job; otherwise false.

@return Returns true if the given process is assigned to the current job instance; otherwise false.

raises

EJwscIWinCallFailedException: can be raised if the call to an winapi function failed.

for the function:

```
function IsProcessInJob(hProcess : TJwProcessHandle) : Boolean;
```

+Add all exceptions that can be raised into the documentation even if they may be raised in used methods. In the last case you can list all used methods instead so the user can see which exception may be raised additionally.

### Class Implementation

+ Don't use private declarations if there isn't a very good reason. First ask your tutor.

+ Stick to this class example

type

```
TJwMyClass = class(...) //see Jw ?
```

```
protected //we use protected, so derived classes can use it
```

```
  fVariable : Integer; //see f the and big letter?
```

```
  //you could add documentation here
```

```
  //but it is
```

```
  procedure SetVariable(const Value : Integer);
```

```
public
```

```
  //document
```

```
  constructor Create...
```

```
  //document
```

```
  destructor Destroy; override;
```

```
  property Variable : Integer read fVariable write SetVariable;
```

```
end;
```

+always use try finally

+Instead you can also use TJwAutoPointer.Wrap