Modernize you Delphi Windows application: use Windows 2000 (and later!) services. (http://www.sandon.it/node/97)

Posted on: Sat, 01/17/2015 - 21:49 **By:** ldsandon

No, the title of this blog post is not a mistake. Delphi, including XE7, only implements services using NT APIs obsoleted since Windows 2000. Windows NT was EOLed in 2004, 2000 in 2010, and XP last year, yet Delphi still doesn't take advantage of the new APIs. What are the advantages? Well, using the "extended" RegisterServiceCtrlHandlerEx() and its HandlerEx() callback, services can receive more and useful notifications (control codes). The new control codes allow to be notified of and handle:

- **Device events**. Service can be notified about device addition/removal without any need of creating a window handle. They can also deny requests.
- **Power events**. Services can be notified of power events (i.e. battery low) and act accordingly.
- **Hardware profile changes**, i.e. a laptop is being docked/undocked.
- **Session changes**, i.e. a user logon/logoffs.
- **Time changes** (only from 7/2008R2)
- **Service trigger events** (only from 7/2008R2)
- **User mode reboots** (only from 8/2012)

There are also some newer event that the old API supports, but Delphi doesn't surface (you could already get them if you override the CustomControl() method):

- **Preshutdown**. Supported only from Vista/2008. Allows a service to postpone shutdown if it needs more time for shutting down properly
- **Param changes**. Service startup parameters are changed, and the service should re-read them.
- **NetBindXXXXX** changes. Same as plug&play notifications, but for services.

Some of this new notifications are very useful to write better behaved services in latest version of Windows. For example, shutdown management changed to ensure faster shutdowns. Terminal server, remoted desktop implies service may need to know user logons and logoffs. Portable devices running on batteries needs power-aware services. And service triggers are useful to avoid always-running services even when they are not needed - just wasting memory and CPU cycles.

In the best BorInCodeDero tradition, the reference OS for the VCL are still Windows 95 and some NT support, especially for whatever is under the hood and people don't see - thereby it may slowly adopt new UI elements, but will be much, much slower to use more current APIs, especially since the driver is to adopt the simplest implementation and avoid to check on which OS version the application is running on, and implement the best support for it, falling back somehow (even raising an exception), if unsupported features are encountered.

Using the new API is not very difficult, but unluckily TService and TServiceApplication were not very well designed with extensibility in mind. Too many fields are private, and some methods that need to be overriden are not virtual. Class helpers could maybe used (as long as there are no scoping issues), or other "hacks", but I preferred to copy the unit, change the class names, and the register both a new TServiceApplication and TService module with the IDE. This doesn't allow for an in-place replacement of TService just changing from what class it inherits from, but anyway there are new events that needs to be surfaced in the property inspector. Let's call the new class TServiceEx.

The key change is to call RegisterServiceCtrlHandlerEx() in the new TServiceEx.Main(). The new API has also a lpContext parameter which is a pointer to user-defined data - and this pointer is then passed to the service HandlerEx function. I used it to pass the TServiceEx instance, and this avoids the "hack" in TService implemetation that uses a global variable to pass the instance to the handler function (which is C callback, and can't be thereby a Delphi method).

Then the new HandlerEx() callback type needs to be implemented. Unlike the old Handler() callback which has no return value (and thereby is implemented as a procedure in Delphi), HandlerEx is a function which returns an error code in a DWORD. It should return ERROR_CALL_NOT_IMPLEMENTED for control codes it doesn't implement, NO_ERROR if everything went OK, or an error code (but check MSDN (http://msdn.microsoft.com/en-us/library/windows/desktop/ms683241(v=vs.85).aspx), because there are some rules to follow). This may become a bit tricky in Delphi because the original TService designer decided to dispatch control codes to the service instance asynchronously using Windows messages. Luckily, the lpContext parameter let the callback access the service instance, and thereby check if a control code is handled or not. In my first implementation, I just use a check if the corresponding event is assigned or not. Maybe it's not the best solution, because a derived class could override the event calling function, and still there is no way to change the return value from within the event (which is called by the thread upon receiving the corresponding message). I'm planning to change the control code dispatching to let the handler function receive the return value by the control code code actually handling it.

The HandlerEx() callback has three parameters more compared to the old Handler() one. One (the last) is the lpContext pointer I already explained. The other two are the lpEventType (DWORD), and lpEventData (LPVOID). Their meaning and values depend on the control code received. Some don't use them (including the ones supported by the old callback), some use both, and some just one. As usual, MSDN is your friend. In the current implementation, where I still use Windows messages as in the old one, I pass them in a record using lParam which was previously unused. The record is allocated dynamically, and a pointer to it is stored in lParam.

The ThreadService message pump frees the memory allocated when the message is received. I modified it also to handle the new control codes and fire new events associated to them, and pass the new parameters when needed. Beware that some control codes are sent to the service only if the service register itself for notifications. For example, RegisterDeviceNotification (http://msdn.microsoft.com/en-us/library/windows/desktop/aa363431(v=vs.85).aspx) still needs to be called to receive such notifications - just in this case it is called passing a service status handle. Previously, this was just a private field, I made it a read only property also to let user code access it.

The last change needed was to modify TServiceApplication (in a new TServiceApplicationEx class) to use the new TServiceEx implementation, there were some casts calling static methods that would have not let the new implementation work otherwise. The finishing touch was to register the new classes using the Open Tools API with the IDE to make them available as a new project type, and a

new module, respectively.

Changes are minimal and I wonder why Embarcadero overlooked them for so long. From a Windows developer perspective, Delphi is lagging more and more behind when it comes to modern Windows development. And asking me why I'm not upgrading to latest releases it's useless. Please, Embarcadero, ask yourself why a $2500 development tools doesn't still support actual basic APIs, and is still stuck in 1995 ones. In twenty years someone could have find some time to "modernize" some basic classes.

Update: some tests on an old Window 2000 system showed that Windows 2000 is pretty picky about what is passed in the dwControlsAccepted field of the SERVICE_STATUS structure. If some unsupported parameter are passed, SetServiceStatus() may return ERROR_INVALID_DATA (later version looks to be more liberal). Thereby is advisable to check the OS version in GetNTControlsAccepted(), and return only supported parameters, for example:

// dwAcceptControl values valid from XP/2003 onwards if TOSVersion.Check(5, 1) then begin if AcceptSessionChange then Result := Result or SERVICE_ACCEPT_SESSIONCHANGE; end;