Published by Apriorit
May 20, 2009

# Interaction between services and applications of user level in Windows Vista



Author:
Yuri Maxiutenko,
Software Developer of Apriorit Inc.

This article is devoted to the question about working with services and applications in Windows Vista. In particular we'll consider how to start an interactive user-level application from the service.This article might be useful for those who deal with the task of organizing the interaction between the service and the application on Windows Vista using both managed and native code.

**Windows Vista, services and desktop**

Before Vista appearance services and user applications in operating systems of Windows family could jointly use session 0. It was possible to easily open windows on the desktop of the current user directly from the service, and also to exchange data between the service and applications by means of window messages. But it became the serious security problem when the whole class of attacks appeared that used windows opened by the services to get access to the services themselves. The mechanism of counteraction to such attacks appeared only in Vista.

In Windows Vista all user logins and logouts are performed in the sessions that differ from the session 0. The possibility of opening windows on the user desktop by the services is very restricted, and if you try to start an application from the service it starts in session 0. Correspondingly if this application is interactive you have to switch to the desktop of session 0. The using of the window messages for data exchange is made considerably harder.

Such security policy is quite defensible. But what if nevertheless you need to start an interactive application on the user desktop from the service? This article describes one of the possible solution variants for this question. Moreover we'll consider several ways of organization of data exchange between services and applications.

Starting interactive applications from the service

As soon as the service and desktop of the current user exist in the different sessions the service will have to "feign" this user to start the interactive applications. To do so we should know the corresponding login name and password or have the LocalSystem account. The second variant is more common so we'll consider it.

So, we create the service with the LocalSystem account. First we should get the token of the current user. In order to do it we:
1) get the list of all terminal sessions;
2) choose the active session;
3) get token of the user logged to the active session;
4) copy the obtained token.
**C++ code** You can see the corresponding code in C++ below.

```
PHANDLE GetCurrentUserToken()
{
    PHANDLE currentToken = 0;
    PHANDLE primaryToken = 0;

    int dwSessionId = 0;
    PHANDLE hUserToken = 0;
    PHANDLE hTokenDup = 0;

    PWTS_SESSION_INFO pSessionInfo = 0;
    DWORD dwCount = 0;

    // Get the list of all terminal sessions    WTSEnumerateSessions(WTS_CURRENT_
    int dataSize = sizeof(WTS_SESSION_INFO);

    // look over obtained list in search of the active session
    for (DWORD i = 0; i < dwCount; ++i)
    {
        WTS_SESSION_INFO si = pSessionInfo<i>;
        if (WTSActive == si.State)
        {
                // If the current session is active - store its ID
            dwSessionId = si.SessionId;
            break;
        }
    }

    // Get token of the logged in user by the active session ID
    BOOL bRet = WTSQueryUserToken(dwSessionId, currentToken);
    if (bRet == false)
    {
        return 0;
    }

    bRet = DuplicateTokenEx(currentToken, TOKEN_ASSIGN_PRIMARY | TOKEN_ALL_ACCESS
    if (bRet == false)
    {
        return 0;
    }
    return primaryToken;
}
```

It sould be mentioned that you can use WTSGetActiveConsoleSessionId() function instead of the looking over the whole list. This function returns the ID of the active session. But when I used it for the practical tasks I discovered that this function doesn't always work while the

variant with looking through the all sessions always gave the correct result.
If there are no logged in users for the current session then the function WTSQueryUserToken() returns FALSE with error code ERROR_NO_TOKEN. Naturally you can't use the code given below in this case.
After we've got the token we can start an application on behalf of the current user. Pay attention that the rights of the application will correspond to the rights of the current user account and not the LocalSystem account.
The code is given below.

```
BOOL Run(const std::string& processPath, const std::string& arguments)
{
    // Get token of the current user
    PHANDLE primaryToken = GetCurrentUserToken();
    if (primaryToken == 0)
    {
        return FALSE;
    }
    STARTUPINFO StartupInfo;
    PROCESS_INFORMATION processInfo;
    StartupInfo.cb = sizeof(STARTUPINFO);

    SECURITY_ATTRIBUTES Security1;
    SECURITY_ATTRIBUTES Security2;

    std::string command = "\"" + processPath + "\"";
    if (arguments.length() != 0)
    {
        command += " " + arguments;
    }

    void* lpEnvironment = NULL;

    // Get all necessary environment variables of logged in user
    // to pass them to the process
    BOOL resultEnv = CreateEnvironmentBlock(&lpEnvironment, primaryToken, FALSE);
    if (resultEnv == 0)
    {
        long nError = GetLastError();
    }

    // Start the process on behalf of the current user
    BOOL result = CreateProcessAsUser(primaryToken, 0, (LPSTR)(command.c_str()),
    CloseHandle(primaryToken);
    return result;
}
```

If the developed software will be used only in Windows Vista and later OSs then you can use CreateProcessWithTokenW() function instead of the CreateProcessAsUser(). It can be called for example in this way:

```
    BOOL result = CreateProcessWithTokenW(primaryToken, LOGON_WITH_PROFILE, 0, (LI
```

Also we must mention that there is a very good article about the launching of the user-level

application from the service with the LocalSystem account privivleges. It is located here: http://www.codeproject.com/KB/vista-security/VistaSessions.aspx