Sometimes it is necessary to create an update - a small installation that copies or replaces one or few files of the application that has already been installed. In most cases, such installation does not require any user interface, or a very simple interface is needed. This article describes how to make an update that checks if main application is installed and replaces several files.

How does it work?

Few words of theory. In gDesigner, go to **Project Options - Application details**, you can find **Application version** field there. The data from this field is a key for making updates. Every installation contains **Application Identifier** - a unique combination of symbols that is stored as a key in HKEY_LOCAL_MACHINESOFTWAREMicrosoftWindowsCurrentVersionUninstall key in Windows registry. On startup the installation checks if a key with the same name as the installation's AppID exists. If it does, the installer reads its **DisplayVersion** parameter and compares it with the version stored in **AppVersion** variable of the project. If the version is greater than the one in the registry - the installation does not show "Select mode" dialog and proceeds with the installation, setting the old version's installation path as default path. The new installation uses old installation's log file and adds records about new files, shortcuts and registry parameters. If it overwrites existing files, no changes are made to the log. So, both of the installation have one uninstaller and the end-user does not have to keep in mind how much upgrades he did to uninstall all of them.

Here is a short to do list to make the update:

1. The basic application must have a version specified in the installation project, i.e. "5.0"
2. The update project should have a version greater than the basic project, for example "5.0.1"
3. Both project should have identical AppID variables.
4. Both project must have the same features, components and plug-ins. Update must have all components even if they are empty. Update must have all plug-ins needed during uninstallation of main applications, although it is possible to avoid it using some tricks that are described further.

Preventing the update to run if proper version of application is not installed

First of all, the update must check if correct version of application is installed. To make the update check for the appropriate version of the basic application, go to **Scripting Actions**, find the **EV_CHECKGLOBALCONDITIONS** in the list of events and press **New Handler** button. Push **Operators** button and select **If operator** from the drop-down list.   Type the following expression:

GetRegParam("HKLMSoftwareMicrosoftWindowsCurrentVersionUninstall%AppID%","DisplayVersion","-1")<>"5.0"

Select **Then** and press **Add Action** button. Type the following expression:

MessageBox("Error", "Application 5.0 is not installed![n]Please install Application 5.0 before installing this update",0,0)

Add one more action:

Return 1

This simple script will prevent the update from installing when a desired version of the application is not installed.

User Interface for the update

The update or a patch might have a very simple user interface, for example two simple dialogs, one asks if the user wants to install the update, another says that the update has successfully been installed. Giving up Custom User interface may save some space:

Using Custom User Interface without bitmapped banners and watermarks - saves 68kb
Not using Custom User Interface - saves additional 125 kb (193 kb total)

To delete all bitmaps and watermarks from the project you have to redesign all dialogs that you plan to use to make them look nice without watermarks and banners and delete all .bmp files from the **Presetup** folder.

How to make a small update without unnecessary plug-ins and interface

Excluding interface is not as simple task as it might seem. I'll try to explain all processes that take place during installation and uninstallation and reasons for all actions that must be taken to exclude user interface.

Uninstall.exe contains information about application's features and components and also contains all plug-ins that were included in installation. During uninstallation the uninstaller scans the log file for items that were installed. For example, here is a part of the log that contains information about a copied file:

```
[Application files.CopyFiles]
   D:Program Files%CompanyName%Main Application Testupdate.gpr=
```

"Application files" is a component id taken from the installation project, CopyFiles is the operation id. If the uninstaller finds records about items from the components that were not included in the installation, the uninstaller ignores them. That is why the update must install files for correct original components.

When an update is installed, it replaces original application's uninstall.exe file, that is why it necessary to have all components, features and plug-ins in the update. But if you really need to make your update as small as possible, you can perform a trick where uninstall.exe is not replaced and therefore it must not have all components and plug-ins.

To leave original uninstall.exe I have created an update that copies its uninstall.exe to another folder and deletes it after installation. I also found out that when uninstall.exe is copied to another folder during installation, it alters UninstallString and ModifyPath registry parameters, so the update must:

1. Save original UninstallString and ModifyPath
2. Install the update having different UninstallPath variable but the same LogFile variable
3. Restore UninstallString and ModifyPath registry parameters
4. Delete the folder with new uninstall.exe

Step by step instructions

First of all, open main application installation file and save it as update. Change Output path and Application version. Delete all files, registry items and shortcuts from all components. If your project has several packages you also have to remove them all but the main package. Now you can add new files to components that must be updated and the update is ready.

If you want to save space then you have to completely remove Custom User Interface and other plug-ins, go to **Advanced Features - Plug-ins** and uncheck CustomUI plug-in. In fact you have to uncheck all plug-ins that are not used during the update. Then go to **Installation Flow - Files** and delete all unnecessary files from the **Presetup** folder. In this case you will be able to show simple message dialogs using **MessageBox** function (like in the script that prevents unwanted installation provided above). Delete empty components and go to Scripting Actions tool.

Create an Event Handler for EV_CHECKGLOBALCONDITIONS that has the following actions:

```
 If MessageBox("%AppName% update to version %AppVersion%", "This will update
%AppName% to version %Version%.[n]Do you want to proceed?",1,1)=1
    then
   if GetRegParam("HKLMSoftwareMicrosoftWindowsCurrentVersionUninstall%AppID%",
"DisplayVersion", "-1")="1.0"
   then
       loop(""; "windowexists("Calculator")"; "")
       if MessageBox("Error","Calculator is running![n]Please shut down calculator to continue
with update",3,2)=1
       then
       continue
       else
       return 1
       else
       MessageBox("Error","%AppName% version 1.0 is not installed or newer version is already
installed[n]Update is aborted",0,0)
       return 1
       else
       return 1
```

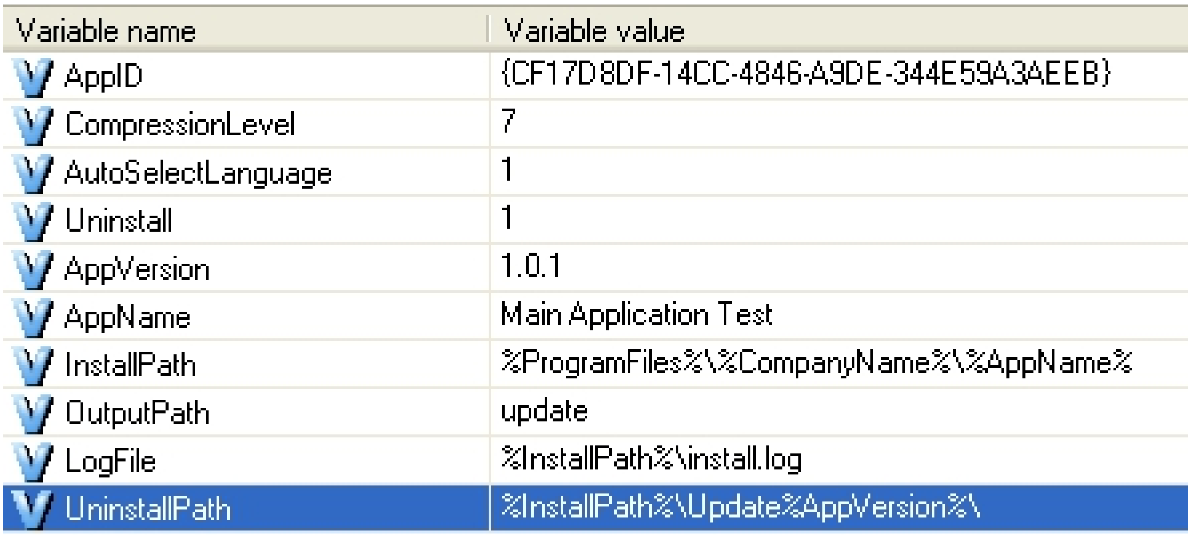
This script shows "Welcome" message. If the user wishes to proceed the script checks if main application is already installed, if it is, the script checks if main app is running (this script checks if calculator is running, just for example). If the app is installed and it is not running then the installation proceeds. Return operators are needed to abort installation, if nonzero value is returned on EV_CHECKGLOBALCONDITIONS event the installation is aborted. To abort installation in other events use AbortInstall function.

Now I must back up UninstallString and ModifyPath as I've mentioned above. To do it I've created a handler for EV_START event:

```
PrevUninstallString :=
GetRegParam("HKLMSoftwareMicrosoftWindowsCurrentVersionUninstall%AppID%",
"UninstallString", "%InstallPath%Uninstall.exe -u")
 PrevModifyString :=
GetRegParam("HKLMSoftwareMicrosoftWindowsCurrentVersionUninstall%AppID%",
"ModifyPath", "%InstallPath%Uninstall.exe")
```

Now the most tricky part of the project. I've changed the value of UninstallPath variable in

Variables tool:



Variable name	Variable value
AppID	{CF17D8DF-14CC-4846-A9DE-344E59A3AEEB}
CompressionLevel	7
AutoSelectLanguage	1
Uninstall	1
AppVersion	1.0.1
AppName	Main Application Test
InstallPath	%ProgramFiles%\%CompanyName%\%AppName%
OutputPath	update
LogFile	%InstallPath%\install.log
UninstallPath	%InstallPath%\Update%AppVersion%\

If I built the update now and run it, it would leave both uninstall.exes in different folders and if the user tries to uninstall the product from Add/Remove applications applet, the last uninstaller would be launched. I want to avoid this, so I've created the following handler for EV_FINISH event:

```
if GetParam(%EventParams%,0)
  then
  SetRegParam("HKLMSoftwareMicrosoftWindowsCurrentVersionUninstall%AppID%",
"UninstallString", %PrevUninstallString%, 0)
  SetRegParam("HKLMSoftwareMicrosoftWindowsCurrentVersionUninstall%AppID%",
"ModifyPath", %PrevModifyString%, 0)
  ClearFolder(%UninstallPath%, "*.*", 1, 1)
  MessageBox("Successful installation", "Congratulations! %AppName% has been successfully
updated to version %AppVersion%.", 0, 0)
  else
  MessageBox("Unsuccessful installation", "%AppName% has not been successfully updated",
0, 0)
```

EventParams variable contains parameter block that is sent by the Engine, the first parameter for this event is a boolean value that indicates wheter update completed successfully.

This script checks if the installation was successful, if it was it restores old registry values, deletes the folder where new uninstall.exe is created and congratulates the user. If the user goes to Add/Remove applet now he sees the version number has been updated and the uninstaller works properly.

This article contains scripts from installation projects, the projects can be downloaded here .

It is impossible to describe all features of such a complex product as Ghost Installer Studio in one article, so I would recommend that you read other articles to find out more about Ghost Installer's exciting features, see features comparison table to know what edition of Ghost Installer meets all your requirements and download Ghost Installer Demo to have the first experience of working with Ghost Installer.