Written by Administrator
Monday, 11 May 2009 09:53 - Last Updated Sunday, 07 June 2009 04:43

One of the most powerful tools that gives ultimate flexibility in Ghost Installer is Scripter plug-in (available in Professional and Enterprise editions). The plug-in allows you to build complex installation scripts written in PascalScript, C++Script, JScript and BasicScript languages. Basically, these languages are slight modifications of Object Pascal (Delphi), C++, Java and Visual Basic. Detail languages specifications can be found here
.

This article explains how to exchange data between scripts created for Scripter plug-in and Ghost Installer and its modules (plug-ins). All examples of scripts used in this article are written in C Script language.

Why was the Scripter plug-in created?

Developers of Ghost Installer 4.0 had one main goal - to make the installer as flexible, as possible, yet the modifications of installations must be much easier than writing a plug-in each time a custom dialog is needed. Three levels of modifications were planned and implemented, the first one is the ability to create custom dialogs for any situation, another is Scripting Actions and the top level is the scripter plug-in. As mentioned above, four scripting languages are available, all you have to do is to write a script in a text file, put it into the %Presetup% folder and use ExecuteScript function. Your script will gain access to installation variables, engine's functions and custom dialogs, used in the installation.

"Hello World!" example

Let us prepare a simple script using C++ Script language, but before that we have to prepare the installation project, that would run our script. First of all, the Scripter plug-in should be enabled in **Advanced features - Plug-ins** (choose Scripter with scripter debugger). Next we have to make an "entry point", a place where the script is started. Actually, the script can be started at any time during the installation, so we will create a new button on the first installation dialog and create the following action for the
OnClick
event for this button:

```
ExecuteScript("c:ScripterTestscript.c")
```

Now create the specified script file (which is c:ScripterTestscript.c in our case) and type the following line in it:

```
{
     MessageBox("Hello World!", "Message",    "0", "0");
      return;
  }
```

Build the installation, start it, press "Run" button and be pleased with your first successful script for Ghost Installer. As you can notice, the "main" procedure does not need any declaration, does not take any parameters and does not return a value. The second thing you can notice here is that the **MessageBox** function is a standard Ghost Installer function for conditions, events and actions, so all Ghost Installer's function are "visible" in scripts. And the last important notice is that all parameters of all functions have a pointer to char type, since Ghost Installer stores all variables and function arguments as strings.

After we have first created the "Hello World!" script, we will discuss the following topics:

Using Ghost Installer's functions.

Almost all has already been told about functions in the "Hello World" example. In conclusion:

- All functions, available in CustomUI events, Scripting Actions and conditions, are available in scripts, use them as if they are already declared. Functions, provided by Ghost Installer plug-ins are also available.
- All parameters of all Ghost Installer's functions have string type.
- Functions return their values as strings.
- Additional information on functions is available in Ghost Installer's help file in **Referenc e - Events - Functions** section
.
- If a function has a variable number of parameters, all of its parameters should be typed in "[" brackets, for example:

```
CallDLL(["user32.dll","MessageBoxA","0","2","0","4","%InstallPath%","4","Test","2","0"])
```

Reading and writing Ghost Installer's variables

All variables, defined inside the script are local variables and are not visible in the project. Variables, defined in the installation project or collected by Ghost Installer (System variables) are available for the scripts. To get a value of Ghost Installer's variable, define a string variable and use get the value of the variable use GetVar function:

```
string a;
  a=GetVar("InstallPath");
```

To change a value of a Ghost Installer's variable, use SetVar function:

SetVar("InstallPath", "%ProgramFiles%%AppName%");

The variable does not need to be declared before calling SetVar function.

Managing parameter blocks.

Since all Ghost Installer functions are available in scripts, working with parameter blocks is just the same as in Scripting Actions or CustomUI events. See next section for an example.

Using Ghost Installer's user interface.

In the "Hello World" example we have used the simplest user interface, but it is possible to use all the power of Custom User Interface, implemented in Ghost Installer. The first thing we need to know is how to show a custom dialog.

First of all, the dialog can be included in the sequence. In this case the dialog should have a condition, that depends on the variable which is changed by the script. This is a way to control the dialog if it is shown in the installation sequence. The script may also change the contents of the dialog using GetProp and SetProp functions, i.e.

SetProp("MyDialog1.CUILabel1.Caption","Something");

If the script must be started from CustomUI events (i.e. when a button is pressed) and the script has to show a dialog, we need to send CM_CUISHOWWINDOW message to the CustomUI plug-in. First of all, the dialog must be created and placed in the "Plug-ins" section. The dialog must have a Dialog ID, we will need its ID to display the dialog later. After the dialog is created, the CustomUI plug-in must be given a command to show it. We need to use SendMessage function and CM_CUISHOWWINDOW. The following script will show a simple survey dialog and process its results(you can download script and the installation project that contains the dialog with the Run button. Make sure you have specified the correct path to

the script in the Execute function in OnClick event for that button. You will have to copy the dialog template to
BinDialog TemplatesMy dialogs
folder in Ghost Installer's installation folder and build the project as described above).

As written in help, we have to fill the InParams block with sufficient parameters for CM_CUISHOWWINDOW message:

```
 string s;
   s=CreateParamsBlock;
   AddParam(s,"DLG_SURVEY"); //dialog identifier
   AddParam(s,"1"); //the dialog should be modal
   string o;
   o=SendMessage("CM_CUISHOWWINDOW",s,"0","0","0");    //this is where the message is
sent
```

We are using a modal dialog, that is why we have to get the result of the SendMessage procedure to get the modal result of this dialog (which button was pressed). Please pay attention, that Ok and Cancel button in this dialog have correct Modal Result properties set.

```
 if (o=="2") MessageBox("Cancel button was pressed","Message","0","0");

 if (o=="1")

 {

 string message1;

 switch (ParseText("%Usage%","0"))
 {

case "1":

message1="You use Ghost Installer less than a year";

 case "2":

 message1="You use Ghost Installer from 1 to 3 years";

 case "4":

 message1="You use Ghost Installer more than 3 years";

 }

 MessageBox(message1,"Message","0","0");

 {

 string k;

 k=GetProp("DLG_SURVEY.CUICheckBox1.Checked");
 k=GetVar("Options");

 int k1;

 k1=StrToInt(k);

 if (k1&&1) Messagebox("You use WebDeploy","Message","0","0");

 if (k1&&2) MessageBox("You use RSA Serial Number      Kit","Message","0","0");

 if (k1&&4) MessageBox("You use Mr.Skinner","Message","0","0");

 if (k1&&8) MessageBox("You use RSA Custom User      Interface","Message","0","0");

 if (k1&&16) MessageBox("You use Scripting Actions","Message","0","0");

 }

 }
```

Summary

Scripter plug-in offers supreme flexibility yet supporting the most popular script languages. Scripts can act as small additions to installation process and be a replacement for plug-ins. Please use the documentation provided for the scripter plug-in when creating scripts.