Automating Build Installs Using AutoIt

Brian Yoss
Brian_Yoss@McAfee.com

Abstract

How do you know which build to use for testing? Where do you look on the network once you know which build to use? Do you use the debug or release version? What install options do you use? How long is this going to take you to get your testing environment up and running? These are all questions that can easily be answered with the help of free scripting tool called AutoIt. This paper will provide you with an understanding of how AutoIt can be used to build an automated GUI utility to help QA acquire the correct build for testing and to get it installed in their environment with a few clicks of the mouse.

On my team at McAfee, it has been a challenge for QA to stay on top of builds delivered from our configuration management system. Depending on the environment, builds could be delivered hourly, daily, weekly, etc. Once QA determines what build they need to use, they need to find it, copy it to their local testing environment, unzip it, and then install it. The actual install process can be daunting and require multiple steps. This can add up to time lost actually testing the product due to the entire build and installation management overhead.

This paper describes possible ideas and processes to be considered when developing an automated utility to produce a streamlined QA testing environment. Further, this type of automation can create testing efficiency for a routine but necessary step, thereby reducing the overall cost to your organization.

Biography

Brian Yoss is a Senior Software QA Engineer at McAfee. He has over 10 years of QA experience and has been at McAfee for 7 years and the QA Lead on McAfee's flagship enterprise security management product for the past 5 years. Brian graduated from Oregon State University with a BS in Business, with an emphasis in Management Information Systems and a minor in Communications.

1 Introduction

I have always had an interest in programming and automation but wasn't quite sure where to get started. I don't have a Computer Science degree, but I know there are tools out there that are easy enough to use without a specific degree that teaches you how to be a programmer. During early 2007, I expressed my interest in automation and was given the opportunity to shadow our automation engineer to see how he did things. While I was learning about some of the processes, concepts, and ideas regarding automation, I was shown a very nice free scripting language called AutoIt. At that time I thought it could be very useful in some of the things we were trying to accomplish, but I didn't have any specific ideas on how I could use it. Fast forward a few years and I finally had some ideas on how I could help improve efficiency and decrease overhead and it involved using AutoIt.

This paper describes my story of looking for QA inefficiencies and building a tool that would be beneficial for not only QA, but other departments within the organization.

Date	Milestone
December 2007	AutoIt automated installer script delivered for Build Verification Test
December 2009	Cluster batch file completed for environment setup
February 2012	ePolicy Orchestrator Install Tool project start
March 2nd, 2012	ePolicy Orchestrator Install Tool Proof of Concept delivered
March 13th, 2012	ePolicy Orchestrator Install Tool Version 0.5 delivered
March 28th, 2012	ePolicy Orchestrator Install Tool Version 1.0 delivered
April 12th, 2012	ePolicy Orchestrator Install Tool Version 1.1 delivered
May 21st, 2012	ePolicy Orchestrator Install Tool Version 1.3 delivered

Table 1: Project Milestone Timelines

2 Build Verification Tests

2.1.1 Problem

Over the past 7 years, I have been involved with testing our enterprise management product called ePolicy Orchestrator (ePO). During December 2007, we were developing a set of automation tests called the Build Verification Tests (BVT) and there was a need to have our product installation automated. The automation tool that we were using at the time didn't have a nice way to automate our product installs and I thought this could be a good opportunity to give AutoIt a chance.

The main pieces that I had to incorporate into the script were:

1) Have the script automatically walk through the installer from start to finish
2) Have the script configurable to allow different settings within the installer

2.1.2 Solution

At a very basic level, AutoIt has a record and playback feature which allowed me to record the exact steps through the install wizard that I needed to use. Recording the steps that I performed allowed me to see how AutoIt inserted the code into the script it created. Once I could see the code, I was able to manipulate it to do exactly what I wanted. After my script was complete, I converted it to an executable file and then we were able to call that file to install our product before our automated BVT was run.

Because of the different environment setups we needed to test against, I needed to make my script configurable to allow different settings during the install process. AutoIt allows configuration files to be

used in conjunction with the script. These files contain different settings that the script reads on execution to allow the script to change what settings are used in the installer.

2.1.3 Drawback

The drawback to this approach was that there were timing issues when clicking through the installer interface. There were a lot of pauses that needed to be manually added to the script in order to address any inconsistencies with clicking through the install wizard. There was also the problem with specific object ID's changing in the installer while we were developing it. If I was looking for a specific object ID to click on and that ID had changed, the installer would hang and would fail. This tool worked for what we needed, but it still required a good amount of overhead and there had to be a better way.

3 Cluster Testing

3.1.1 Problem

ePO is a product that is cluster aware and we support failover capabilities. I was given the task to ensure our product functioned correctly in a clustered environment. Building a clustered environment for testing is a very daunting task. Assuming I already have the virtual machines available for use, some of the tasks involved are:

1) Setting up at least 2 systems to act as active and passive nodes. This typically takes about 30-45 minutes for each node.
2) Setting up a storage server for the application and cluster data which takes around an hour.
3) Setting up a remote SQL server which can also take about an hour.
4) Installing the Cluster Service in the environment which takes about 30 minutes.
5) Installing ePO which takes about 30 minutes.

Because of the amount of setup that is involved when setting up a clustered environment, I thought of ways that I could automate the process in order to lower the amount of overhead involved. The initial time for setup would be the highest cost, but once it is all configured properly, the future overhead is very low.

During December of 2009, I sat down and determined that the main pieces that I had to incorporate into the script were:

1) Allow the user to specify a build, package, and build type to install
2) Have the script revert VM snapshots and power them on
3) Have the script configure the cluster settings
4) Have the script find the build specified and copy it to my test system
5) Have the script unzip the build
6) Have the script install the build

3.1.2 Solution

After some initial research, I discovered that the majority of the tasks I was trying to accomplish could be completed via a command line. I was using virtual machines that I could call specific snapshots, power on, and power off from the command line. Our product could be installed silently via a command line. I could even configure my cluster environment using the command line. I had some experience in the past with building batch files, and I thought this would be something I could start with and build upon.

I was able to successfully use my batch file to build my cluster environment whenever I needed it. I cut down my environment setup time from about 4 hours to 45 minutes. There was a lot of interest in how I

created my batch file as soon as people heard about the setup efficiency because it could be used for any environment and not only for a cluster environment.

3.1.3 Drawback

I created documentation for others on the team to be able to create their own batch file. However, there was a lot of initial setup involved to get the batch file to run smoothly and it wasn't exactly user friendly. People wanted something easier.

4 ePO Install Tool (Proof Of Concept)

4.1.1 Problem

During Q1 of 2012, McAfee started tasking employees with coming up with "innovation" ideas within the organization. I thought this was a perfect time to take what I had done in my batch file and use AutoIt to build a nice Graphical User Interface (GUI) that was very user friendly so anyone could use it. I knew I had to start with a proof of concept (POC) and build it out from there to see what I could do. I wasn't sure what the end result would be, but I was up for the challenge.

The main pieces that I needed to incorporate into the tool were:

1) Allow the user to specify a build to install
2) Allow the user to specify any network credentials that are needed for the install
3) Allow the user to specify if they wanted to install the product using our packaged SQL Express or point to an existing SQL Server
4) Allow the user to specify the admin password for the product being installed
5) Mask any password information that the user enters
6) Only allow the install to start after all required fields are filled out
7) Have the tool automate the process of copying the build locally
8) Have the tool automate the process of unzipping the build
9) Have the tool automate the process of installing any pre-requisites
10) Have the tool automate the process of installing the build

4.1.2 Solution

Since I hadn't used the GUI piece of AutoIt before, I spent some initial time researching the API to see exactly how to create the GUI. AutoIt has a wonderful set of web pages that have a ton of information and examples of how to create some amazing tools.

Once I figured out how to create a baseline GUI, I then had to determine what functions I needed to use to add the pieces of functionality that I determined were needed. I knew there were a lot of different configuration options that I could include, but I wanted to keep things simple at first to see if this was a feasible project.

Because there are different versions of the product that the tool needed to accommodate, I was able to keep the version specific information on their own tabs. The information that was generic across version was kept consistent for each version that was selected.

When the tool launches, it reads the current environment such as domain, user, and machine name and adds this information as default values to help the user. Each one of the pre-populated values can be changed based on the user's needs. Below is a screenshot of my POC and I sent this out to a select few within QA to get feedback and to address any issues that they found.

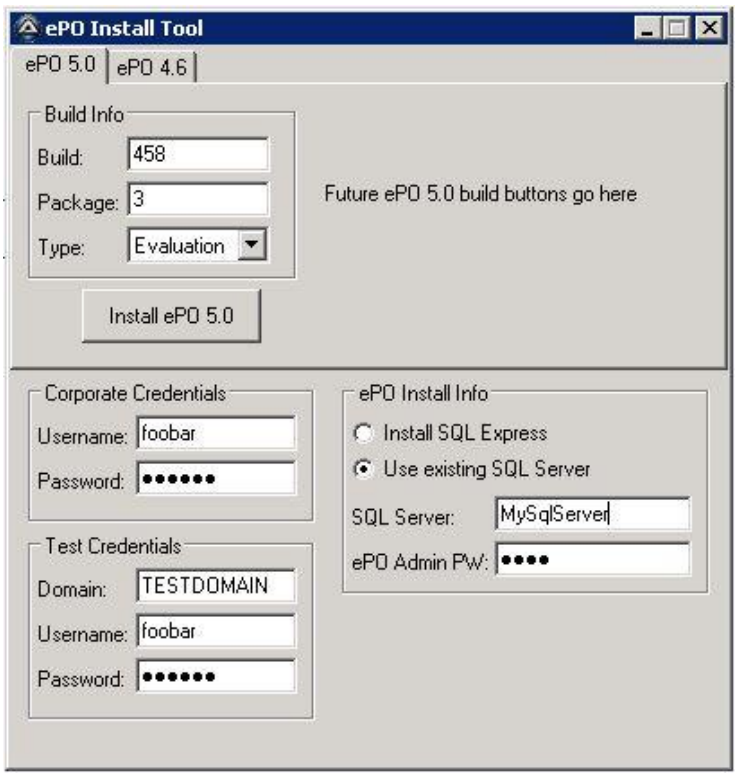

Figure 1: ePO Install Tool POC – 5.0 Tab

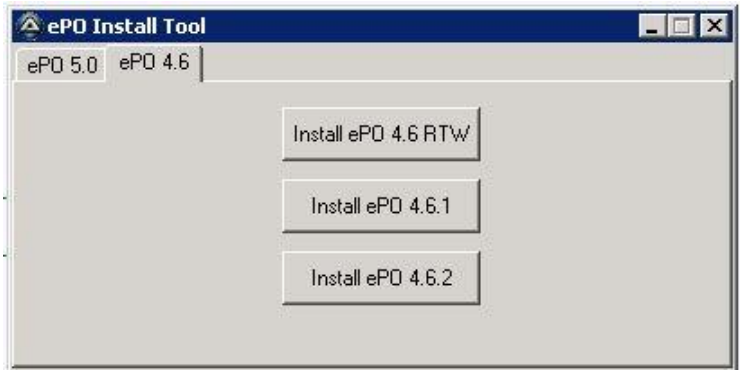

Figure 2: ePO Install Tool POC – 4.6 Tab

4.1.3 Drawback

This was a good start, but there were still a lot of features that could be added or improved upon. At this point I knew that my idea was possible, but I needed to add more pieces to really make this useful for others.

One of the main pieces of feedback that I received was to add a list of possible builds to install instead of needing to know the exact build, package, and type.

5 ePO Install Tool (Version 0.5)

5.1.1 Updates

Based on the feedback that I received from my POC, I investigated a way to provide a list of possible builds to install. McAfee uses a Configuration Management DB (eCM) that stores all of the information regarding our builds for every product. After some research, I discovered that AutoIt has built in functions to connect to SQL databases. I just needed to get the credentials to the SQL DB and then I could run a query against our ePO DB to return the results of our builds. After I had the results of our builds, I could pass that info into an array and populate a list box in the GUI to display the last 5 builds available in eCM. With the addition of a list box of possible builds to use, users could select a build to install or could still manually enter the build, package, and type if they chose to do so.

I also added a File and Help menu system that allowed users to exit the tool and to use help to navigate to an internal webpage that included information about all of the different pieces of functionality. After these updates were made to the tool, it was sent out to a wider audience that included QA, Development, and Technical Publications.

Below is a screenshot that displays the menu system and what the list box looks like so that users can see the available builds and their status. Selecting a build automatically populates the build, package, and type fields in order to install the build.

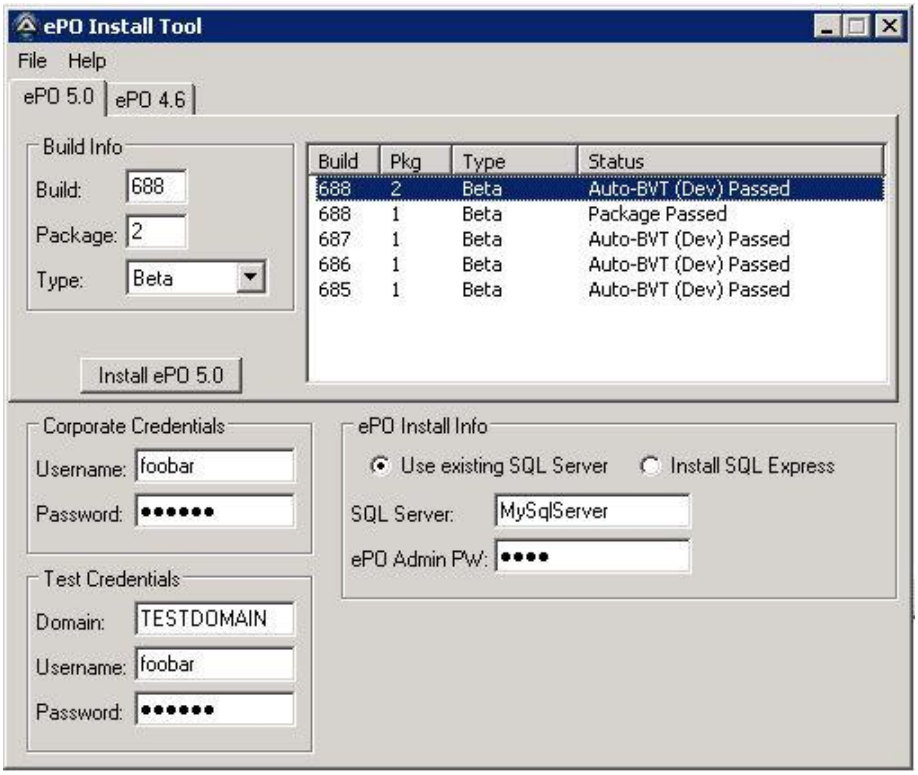

Figure 3: ePO Install Tool – Version 0.5

6 ePO Install Tool (Version 1.0)

6.1.1 Updates

During the release of Version 0.5, I started to receive a lot of feedback from users. QA, Development, Technical Publications, among others were all using the tool and they wanted to see more functionality. I even had people in the organization offer to help with updates since this wasn't my primary job and it was becoming something that required a lot of time to incorporate all of the updates that people wanted to see. I knew that there were still a lot of things that could be added to the tool, it was just a matter of finding the time to get it done. Some of the feature requests were:

1) I want to see more than the last 5 builds.
2) I want to install an Agent Handler too. Agent Handlers are a separate install that tie directly with ePO.
3) I want to be able to install builds with more versions of ePO.
4) I want to be able to provide the port that my SQL Server is using.
5) I want to be able to provide the admin name for my ePO install.
6) I want to be able to enter a license key.
7) I want to be able to provide the install path.

Based on this feedback, I started to update the tool. The first request was really easy because I just needed to change the SQL query from returning 5 to a higher number. I changed this to the last 20 builds, which I felt would suffice.

ePO has a piece of functionality called an Agent Handler, which is a separate installer. Users wanted to be able to use this tool to install an Agent Handler too. In order to install an Agent Handler, the user needs to provide the name of the ePO server that it will connect to. It also needs to know the admin name and password that ePO is using. These fields were added to the bottom of the tool in its own section labeled "AH Install Info."

During ePO development, there are a lot of different versions being developed and supported. Users wanted to be able to install all different versions of ePO. Tabs worked really well in this situation, so I added a new tab for each version that was being supported. Since ePO 4.6.3 was actively under development, I was able to use the SQL query to the eCM DB to pull the latest builds for that version as well.

Users also wanted to be able to specify the port that their SQL Servers were using. When the tool is launched, it defaults to the standard SQL port that we use. However, this port could be different based on their environment and users can change it for their specific needs.

In the previous versions of the tool, it only allowed the user to provide an admin password for the ePO install because that is the minimum info that is required for the admin. When the tool is launched, it defaults to the standard admin username that we use and users can now change that to what they need.

After an ePO install, users can enter a license key in ePO to provide access to different pieces of functionality. Instead of making users enter the license key manually after an install, they wanted to provide this info in the tool so it was done automatically. By default, the tool enters one of our standard testing license keys, but can also be changed to the users liking.

Because of all the different environment configurations that we need to test, users want to be able to install ePO to a specific path that they entered instead of using the default path. When the tool is launched, the default install path is provided and users can change it to their needs.

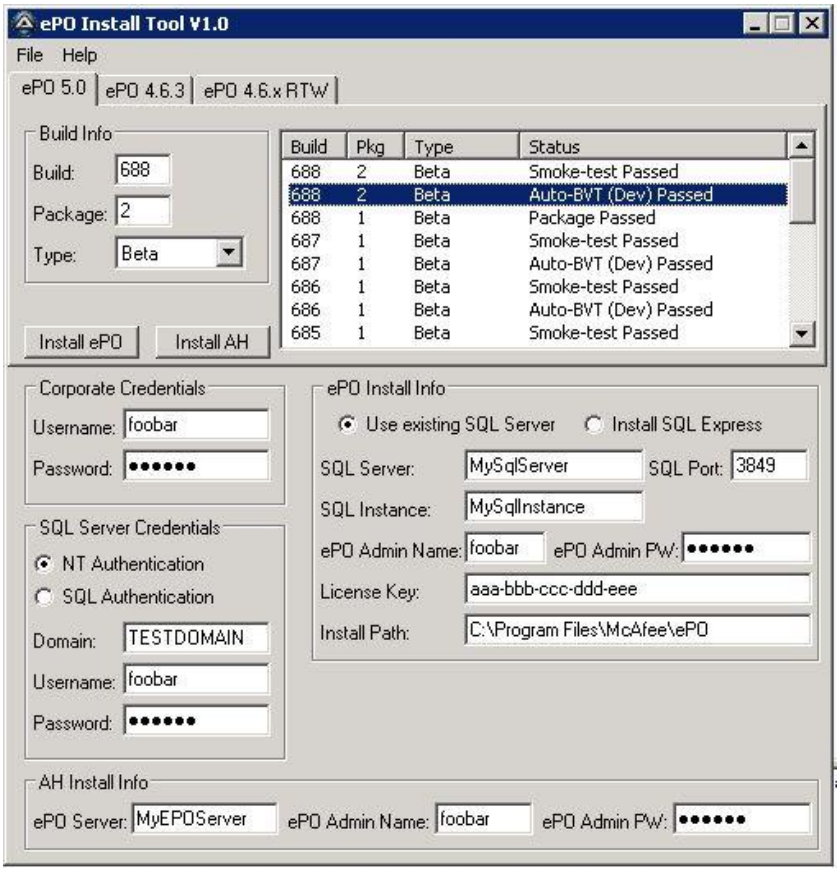

Figure 4: ePO Install Tool 5.0 Tab – Version 1.0

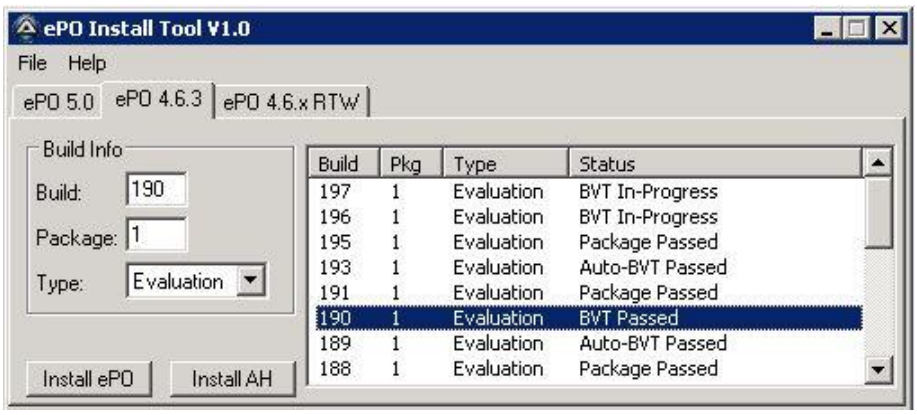

Figure 5: ePO Install Tool 4.6.3 – Version 1.0

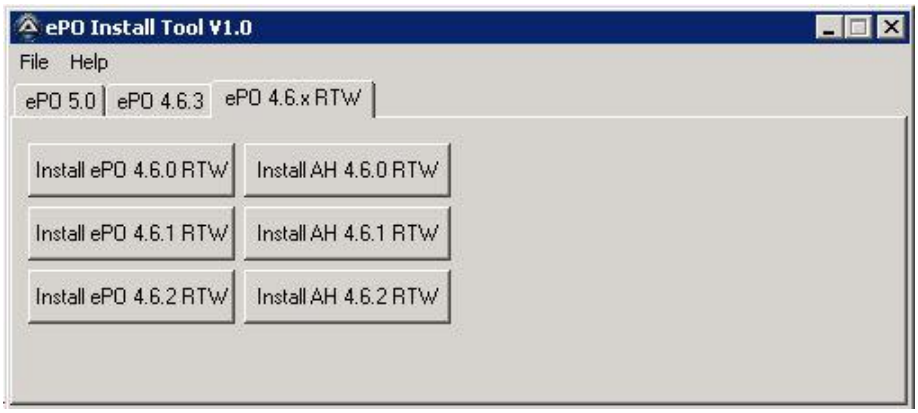

Figure 6: ePO Install Tool 4.6.x – Version 1.0

7 ePO Install Tool (Version 1.1)

7.1.1 Updates

Things were coming along pretty well and the tool was actively being used by multiple users across departments. I was still getting a lot of feedback, including:

1) Provide all of the port info for an ePO install so I can specify what I want and verify the ports aren't already in use before the install starts.
2) Provide more feedback when the install is occurring so I know what is happening.
3) Give me a way to just launch the tool and click "Install" without needing to enter all those values.
4) For debugging purposes if something goes wrong, I want to see a log file of what happened.

ePO uses a number of different ports during the install. These are all configurable in our installer, so users wanted that functionality in the tool as well. I added port info in its own section labeled "ePO Port Info" and when the tool launches, it defaults to our standard ports. The users can change these ports to anything they would like for testing purposes. I was able to find a function on the AutoIt forums that provided port checking and I used that to validate that the ports specified in the tool were not being used on the system.

Once the user clicks the "Install" button, there wasn't a lot of information displayed to the user as to what exactly was happening. Because of this, I added a status bar at the bottom of the tool that accurately displayed a status at each step of the install process after clicking "Install." When the install is completed, a dialog box is also presented to the user letting them know if the install was successful or not.

Users still wanted this process to be easier. The tool was very useful, but there were a lot of fields now that possibly had to be filled in depending on the environment that was being used. Users wanted a way to just launch the tool and click "Install." I remembered when I built the automated script for our BVT's that I used a configuration file that the script would use upon running it. I wanted to use this same concept and allow users to have a configuration file that the tool could read that would automatically input all values. All the user would need to do is select the configuration file, select the build they wanted, and click "Install." Under the File menu, I added an option of "Load Configuration File…" and "Save Configuration File…" When a user has launched the tool and input all the field values according to their environment, they would save the configuration file. The next time they used the tool, they could load their configuration file and all of the values they had saved previously would load into the tool. This would allow users to have multiple configuration files for each one of their different environments.

Sometimes the install would have a failure and users wanted to see a log file during the install so we could determine what went wrong. I added logging to the tool so that a log file was created in the same

directory as the tool and it would report the exact steps that were occurring. Based on the contents of the log file, you could determine where the install failed.

Below is a screenshot of the newly added port info and status bar.

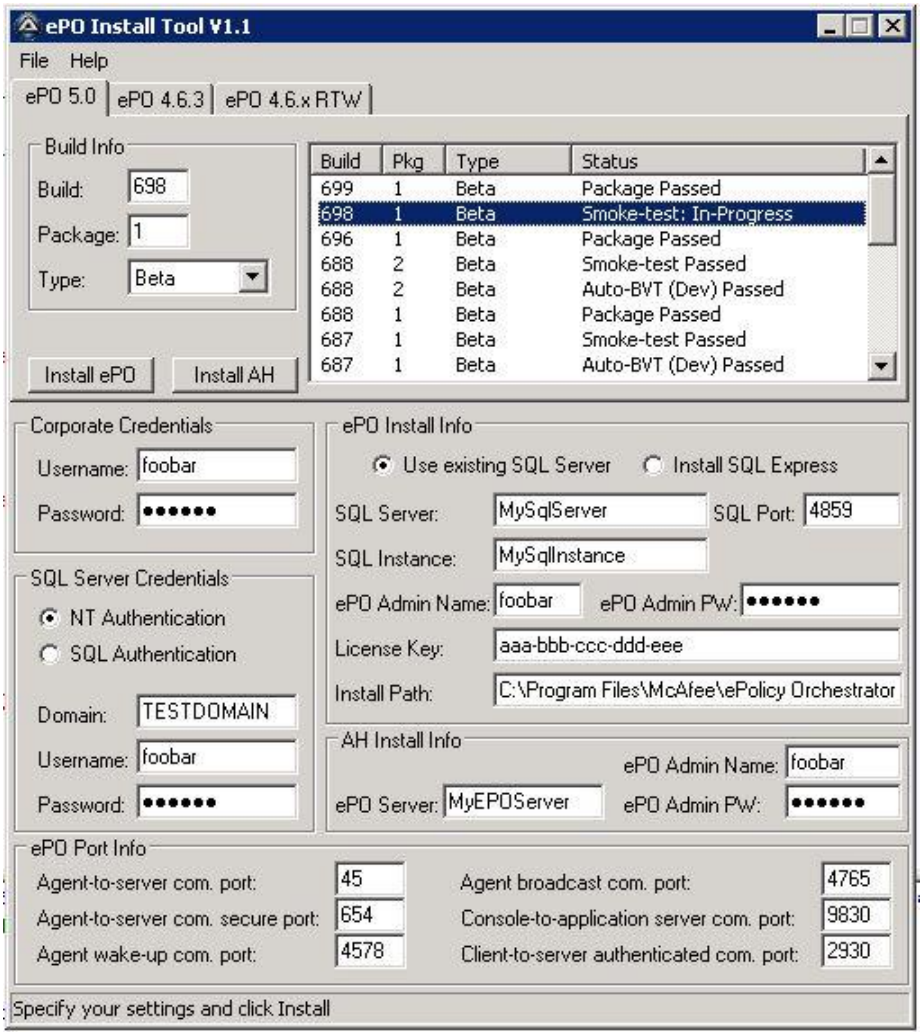

Figure 7: ePO Install Tool – Version 1.1

8 ePO Install Tool (Version 1.3)

8.1.1 Updates

At this point, the tool was very stable and was widely in use. There were approximately 20 people using the tool by this time and that number has been growing steadily. There were still things that people wanted to see added, but the majority of features had been integrated. Our Performance QA Engineer asked if he could help me work on the tool because he wanted to integrate it into his performance testing. Since I knew he had one of those CS degrees, I gladly accepted his offer to see what he could do. There wasn't a lot of UI changes that he made, but he took a lot of the logic in the code that I had written and dramatically increased the efficiency of execution.

We were able to remove the "Corporate Credentials" section because it wasn't really needed on a per user basis. Those credentials were primarily used for copying the build off of the network and we had a

general corporate account that we could use for all users. We now provide those credentials in the code instead of requiring the user to enter them.

Below is a screenshot of Version 1.3, which is currently being used.

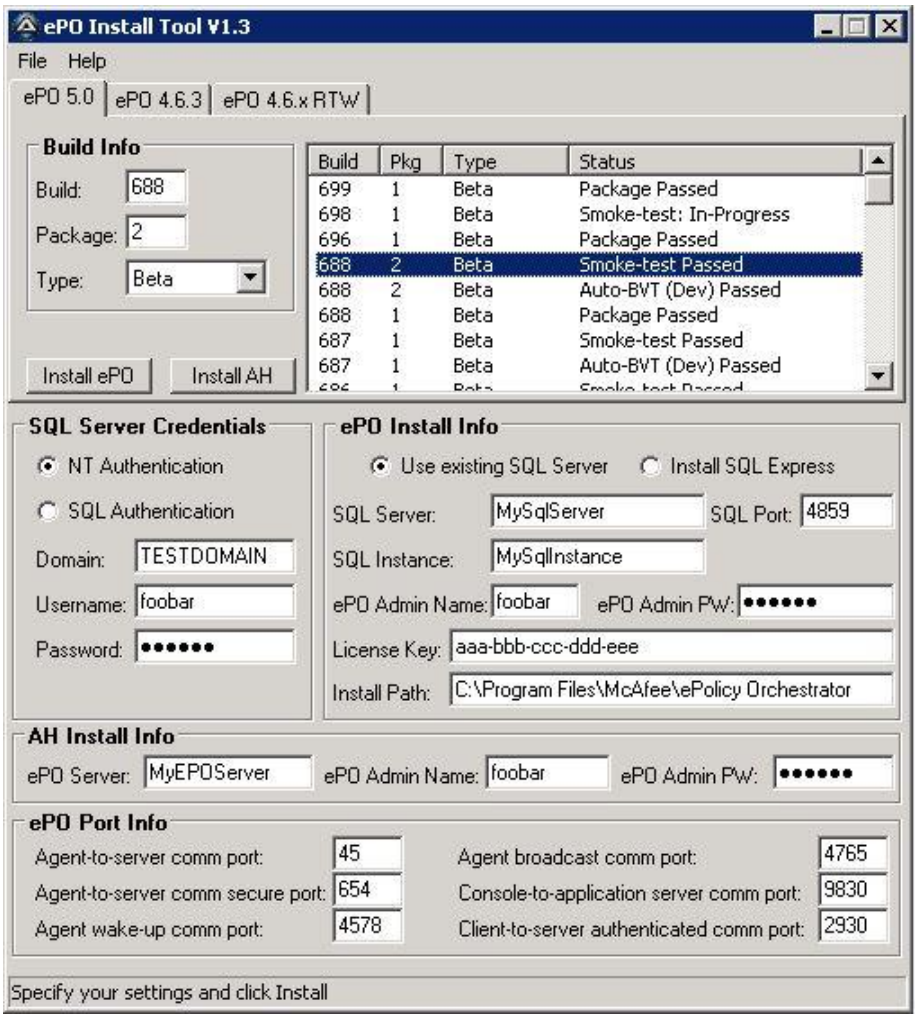

Figure 8: ePO Install Tool – Version 1.3

9 Future Updates

There are still a lot of feature requests and ideas that are coming in for the tool. It's just a matter of finding the time to get these added. Some of the features that users want to see are:

1) I want to be able to run the install on a remote system instead of only on my local system.
2) I want to use the tool to perform an upgrade instead of only a clean install.
3) I want the install to check for an existing DB on my SQL Server before installing.
4) I want more options and other product installs…

Currently the user is required to run the tool on their local system. Users would like the ability to point to a remote system and install ePO to that remote system.

The tool only supports clean installs at this time. Eventually I would like to add functionality to perform an upgrade as well.

If you attempt to install ePO using a SQL Server that already contains a DB with the same name from a previous install, the install will fail. Users are required to verify that there isn't an existing DB before the do an install. I would like to check for an existing DB before the install started. If there was an existing DB, I would prompt the user and give them the option to delete it.

I could go on and on regarding the features the users want to see. The possibilities are almost endless.

10 Summary and Conclusions

Trying to get builds installed and configured in your environment can be a daunting task, especially when a new build could be delivered as often as every few hours. Something that shouldn't take a long time can end up being a large time sync and can take away from valuable time actually performing your daily activities. AutoIt is a very powerful scripting language which can help automate some of the more mundane and repetitive tasks. It doesn't take a Computer Science degree to build a tool like this that will help increase efficiency and reduce overall cost to your organization.

References

AutoIt: http://www.autoitscript.com/site/autoit/