The following paper was originally published in the

Proceedings of LISA-NT:
The 2nd Large Installation System Administration of Windows NT Conference

Seattle, Washington, USA, July 16–17, 1999

STATE-DRIVEN SOFTWARE INSTALLATION FOR WINDOWS NT

Martin Sjölin

State Driven Software Installation for Windows NT

Martin Sjölin
Warburg Dillon Read (WDR)[*]
P.O. Box, 8098 Zürich, Switzerland
martin.sjoelin@wdr.com

Abstract

We have implemented a state driven installation mechanism to simplify the installation of software under Windows NT. We have a "central configuration database" which defines the target state of the machines, e.g. a declarative definition instead of an operational definition. We describe the procedure used to install the packages and some related issues concerning software delivery; software configuration; and the actual software installation for workstations components and user components. Partial details of our implementation of System V packaging is included in the Appendix.

1 Introduction

To make the installation of software in the desktop computing environment of UBS, we have implemented a state driven software installation mechanism. We are using Microsoft NT workstation as the standard desktop with Netware servers or NT servers as the associated file and print (f&p) servers or as home servers for user data. The tools described have been productive use since the late 1997.

We have selected to "lock" down the desktop and the machine for the normal users. And having a standard set of applications which are "packaged" using our own packaging format similar to the "System V" packaging [SysV]. This format is used for delivery, configuration and installation of the applications (please see the appendix for more details). The packages are delivered via CD or SMS [SMS1.2] to servers on the local LAN in the branch office. Then is the software installed on the f&p servers and/or workstations (clients).

To ensure that all workstations have a standard set of applications installed, we store the configuration of each workstation in a "centralized database" [1] (this is not an inventory database). The configuration data defines pro workstation which packages should be installed, the installation mode, and the installation order between the packages. The users of the workstation is restricted to the set of packages listed in the configuration database for the workstation.

At well defined times, a NT service running on all NT workstations and NT servers wakes up or is awakened. The service compares the current state of the machine (which packages are actually installed) against the state defined the configuration data. The service then performs the necessary actions to reach the state as defined in the configuration data.

When the users login into "their" workstations, we compare the user components already installed into the user's profile (including home drive) against the set of packages installed on the workstation itself. We compute the difference between the state of the workstation and user's profile, and then perform the necessary actions to update the user's profile to the state of workstation[2]. We name this *flex seating* since the users can flex or change between the different workstations which are similarly configured, having access to all the standard application via their profile[3].

[*]The work described was done while the author was at UBS AG, Private and Corporate Consumer Division, between 1996 and 1998.

[1]The database is stored securely on a server, either as flat files, ini files, or in a RDBMS.

[2]In reality, it is more complicated, but the scope of the paper does not allow us to elaborate on the software authorization mechanism.

[3]To allow full roaming between different branch offices or different resource domains, we need to add support for on demand package installation.

For the NT servers, the same process applies as for the NT workstations, but with a single extension - the NT servers must also handle packages which are exported to the clients, e.g. dictionaries for Microsoft Office. Compare this against the packages which are installed for the NT server itself, such as SNMP, backup, virus scanner etc.

As seen from the above description, we have a declarative definition of what should be installed on each machine and by extension which application users have access to. By have the state of the machine defined in a configuration database, it is very easy to recreate the state of the machine after a crash[4]. Compare this with more "normal" operational definition where the state of the machine is defined by the set of application installed on the machine, possible together stored a central inventory of the machines.

2 Computing Environment

We support user authentication against a NT domain with home directory and profiles stored on a NT server, or against Novell's NDS where the user profile and home directory is stored on a local Netware 4.x server. For the typical environment in a branch office (the listed server are logical servers, often there will be a single physical server), we have:

- *Clients*, the workstation or desktop machines, which are all running Microsoft NT 4.0. All client have an associated shared application server.
- *Shared Application Server* which is where common components like dictionaries, templates, and seldom used applications are stored. This is often the print server for the associated clients. A few standard SMB shares are available for the client.
- *Home Server* is where the user home directory is mapped and stored together with the NT 4 user profile.
- *Package Server* stores the packages to be installed. Exports a SMB share with the *package spool area*. For a SMS distribution server we also have the standard SMS package share. For NT servers, we distribute package via SMS and for Netware server we distribute packages via monthly CDs. Or via HTTP or FTP in emergency cases.
- *Configuration Data Server* stores the "configuration database" as `ini` style file for the machines.

[4]We use the same mechanism for the initial setup of the machine by providing a small set of bootstrap tools in the OEM directory of the Microsoft unattended setup, and then invoking our installation process.

3 Implementation

The software installation (including both removal old versions and additions of new versions of software) can logically be split into the installation of:

- F&P servers components is the shared context of packages. These are made available for the workstation from its associated f&p server via SMB shares. For NT servers, the software installation is done by the "EUP Installer Service"(`eupsrv`).
- Workstation components includes the workstation context of a package for a *shared* mode installation, but can also include the shared context and the workstation context for the *local* mode installation. As for the NT server, the software is installed by the "EUP Installer Service"(`eupsrv`).
- User components are only the user contexts of a package, installed by by the `euplogon` program at user login time.

We treat the installation of server components and workstation components as a single case. From the actual calculations the server case is an extension of the workstation case to handle the shared context exported to the clients via the SMB shares.

For the installation of the server and workstation components, we need the configuration data describing the wanted or target state of the machine. In the configuration data for the machines are stored: the list of packages to be installed, their installation mode (local, shared, or exported), the location of the package spool, and control parameters for `eupsrv`. The installation order of the packages is the order of the packages in the configuration data.

3.1 Machine Components

The first step for the **eupsrv** is to determine if any users are logged in - we user several methods: a GINA [GINA], enumeration of open desktops, and a flag set by the **euplogon** at user login. If an user is present, we either present a warning dialog asking the user to logout as soon as possible, or we will retry within a pre configured time limit (defaults to 4 hours). Once the **eupsrv** can detected no users, it optionally login into the associated shared application server using the credentials provided in the registry configuration. This is to get access to the configuration data, to the shared application server, and to the package server.

The initial step is to determine the set of packages already installed on the machine, state (installed or removed etc.), mode (local, shared or exported), the installation order, and who installed them and when. By scanning the pkginfo directory, for a NT workstation and a NT server: `%SystemRoot%\Config\PkgInfo\`. And for the NT server the list of exported packages: `\\server\PkgInfo\`.

For each package we read the installed package's **pkginfo.ini** and **pkgvars.ini** files to retrieve the installation time, the shared application server, installation status (successful installed, partially installed, or removed):

```
[Status]
PSTAMP=MARTIN980226113655
UserId=SYSTEM
Status=Successfully Installed
Date=1999.11.11:23:05:00
```

The shared application server associated with each package is retrieved by reading the value from the **pkgvars.ini** file. The installation mode is *shared* if no shared context have been locally installed. If a shared context and workstation context have been locally installed, the package have been *locally* installed.

Thus, we have determined the set of current packages installed on the machine:

$$CurrentState = set\ of\ packages\ already\ installed \quad (1)$$

We connect to the configuration database to retrieve the target state, which is an ordered list of packages with their installation mode:

$$TargetState = set\ of\ packages\ in\ the\ configuration \quad (2)$$

Having *CurrentState* and *TargetState*, we compute the set of package necessary to remove from the machine, by taking all packages not present in the *TargetState* but currently found on the machine, *CurrentState*:

$$Pkgs2Remove = CurrentState \setminus TargetState \quad (3)$$

By default, we filter out package which have not been installed by the service from the *Pkgs2Remove* set, since removing the Emacs package which a developer have installed for his own use is not acceptable.

The next step is to compute the set of packages which we must install on the machine to reach *TargetState*, which is the set of all package present in the *TargetState* but not in the *CurrentState*:

$$Pkgs2Install = TargetState \setminus CurrentState \quad (4)$$

Notice that two entries when comparing are only considered equal if the following conditions are met:

1. Same package name (**ubsperl_5_un_1.4**),
2. Same installation mode (shared, local or exported),
3. Same shared application server for *shared* installation mode, and
4. Already installed package is installed successfully.

Once we have computed the *Pkgs2Install* and *Pkgs2Remove* sets, we order the *Pkgs2Remove* in the reverse installation time to avoid any problems with install time or run-time dependencies. The *Pkgs2Install* set should will the same order as specified in the configuration data. If both *Pkgs2Remove* and *Pkgs2Install* sets are empty we are finished.

Before we start the actual package addition or package removal, we verify for all package in the *Pkgs2Install* set that the package is actually present on the package server. We also verify that for shared mode installation that the correct version of the shared context of the package is installed on the the shared application server. If not both of these conditions are fulfilled, we will ignore the package during the actual installation.

We start traversing the *Pkgs2Remove* set and remove the already installed packages by calling `pkgrm`. For the shared mode, we remove the workstation context (which is the only context installed). And for the local mode, we first remove the workstation context followed by the shared context.

The following step is to process the *Pkgs2Install* set and to install the new packages by invoking the `pkgadd` for each package. The major parameters are: package name, shared application server, and path to package spool area. For locally installed packages, we first install the shared context and then the workstation context.

The `eupsrv` invokes *pre-* and *post-* scripts stored on the machine in a secure location and stored on the shared application server before the enumeration of the installed package as well as after adding the last package. This enable packagers or the local supervisor to modify the state of the machines and have to influence what are installed.

Last, for a package which requires an immediate reboot, as specified by the restart flag in the `pkginfo.ini`, the `eupsrv` will force a reboot of the machine once the package have been removed (or added). For packages which requires a reboot before becoming operational, the `eupsrv` will reboot after the last action (install or remove) has been done. The `eupsrv` will continue operations after the reboot, either being restarted by the SMS PCM (Package Command Manager) or by itself.

3.2 User Components

When the user login to a NT workstation, the `euplogon` program is invoked - we have replaced the standard `UserInit` registry value. The `euplogon` program compares the list of packages already installed on the machine (by scanning the machine's pkginfo directory) against the list of user contexts already installed to the user profile and home drive (by scanning the pkginfo directory stored on the user home drive and checking the cached values in the registry).

Using roughly the same computation as described in the previous section, the program determine the set of user contexts to remove and the set of user contexts to add to ensure that the user have the same set of packages as already installed on the machine.

One minor change is that we only remove a user context if the current machine where the user is logged in to is the same machine where we originally installed the user context. We avoid removing a package when an user temporarily login into another workstation than the her normal workstation. Instead, we hide any shortcut in the user program menus by setting the `HIDDEN` bit on the shortcut.

4 SMS Integration

Since one year, we support software distribution via SMS and also initiate software installation via SMS for the "pure" NT environment. This integration caused a number of changes or improvement in the `eupsrv`.

4.1 Push versus Poll

When we control software installation using SMS, we would like to initiate the software installation via a standard SMS job. This changes the operational mode from "poll" (check if configuration data have changes at regular intervals) to "push" (start now and verify if the state of the machine matches the state as defined the configuration data).

The standard operational mode of the `eupsrv` in the initial release was to poll the configuration data (a `ini` file) every fourth hours when a user was not logged in. To avoid re reading the file when it not have changed, we cache the last modification date and size in the `eupsrv` and compare those attribute to determine if the file have changed.

We have the `PCM` installed on all NT workstations, NT servers, and SMS distribution servers as a service. We extended the `eupsrv` to be started from the `PCM` and no longer polling the configuration data at regular interval. When called from a SMS job, the `eupsrv` service is started from the `PCM`. The command line executable blocks until the `eupsrv` service returns with a status code before it exits (and thus blocks the PCM and the current SMS job).

The `eupsrv` was extend with three command line option to support:

- `eupsrv -run` which verify the state of the machine against the state as defined the configuration data. This command is invoked as part of a SMS job to force an update of a machine.
- `eupsrv -pkgadd` *package-id*,... to enable the installation of one or more package from a SMS job, *if* the packages are included in the configuration data for the machine. The installation mode is read from the configuration data.
- `eupsrv -pkgrm` *SMS-ID*,... to enable the removal of one or more packages from the target machine. No check is done for run time dependencies, so this can break an existing installation.

Some complication arise in the handling of packages which requires a reboot or who should restart the `eupsrv` after the reboot. The simple solution was to stop the SMS `PCM` service when a reboot was required, and having `eupsrv` forcing a reboot of the machine. After the reboot the `PCM` detects that the SMS job was not finished, and retries SMS job including the `eupsrv` command.

4.2 SMS Software Delivery

One major problem with SMS 1.2 is that the actual software distribution is not atomic to the distribution servers. By atomic delivery, we mean that either is the package on the SMS share to 100% or to 0%, not partially present.

To all SMS distribution jobs, we added a small batch script which created a flag file, in the root of the package directory once the package was delivered to the distribution server. The second step was to extend the `eupsrv` to verify if the package was fully delivered to the share by checking for the flag file.

4.3 Shared Contexts Coordination

For packages which are installed in shared mode on the workstations, we must ensure that the shared context is present on the associated shared application server and also that the correct version of the package is present. This was added as part of the SMS extensions to ensure that SMS jobs sent directly to the workstations did not try install the workstation context of the package before the shared context was present on the associated f&p server.

5 The Good, the Bad and the Ugly

We have learned a few lessons during the last years during the development of the installation mechanism, especially issues which crept up during the integration with SMS. What follows is a partial lists of points.

We have discovered bugs in both Microsoft Networking code and the Netware client for NT. The network and security issues to get the `eupsrv` working correctly have been causing headaches.

The "Poll" mode sounds good and looks good, but the local supervisor needs more control over when an software installation is started, which workstations should be done, and a mean to reduce the load on the network and/or the server. We solved this by adding a common configuration file on the local Netware server, but ...

Under Netware 4.x, we need to authenticate against NDS before we can read files on the server, e.g. the package or configuration files. In the case of the configuration file above, even though we had limited the number of parallel updates to, say 5, we overloaded the NDS authentication server when all the workstations tried to read the configuration file. For the Netware environment, a possible solution would to use a TCP server for common configuration data, or start the `eupsrv` via Netware Workstation Manager.

The current general policy for the installation of a new version of a package is first to completely remove the old version and then install the new version. But most of the new versions of packages are incremental improvement or small changes to the components. The current approach causes a lot of extra network traffic by coping data from the package server which to a large extend was already present on the target machine. Either overwrite functionality or a delta package mechanism to reduce the network traffic as well as the installation time should be considered. The current "distributed" database of installed components, `pkgstat.dat`, in each package's pkginfo directory pro context, should be centralized to easy the implementation of overwrite packages.

Centralize the configuration data in a central repository (database) and distribute it using LDAP [LDAP] or similar. By extending the initial design with new configuration files on the shared application server, we added more and more configuration data in several location instead of going for a unified interface. With the new release using the *EUP Values* we have initiated the centralization and collection of the configuration data into a single location and single interface.

During the last year, we have put a lot of work into the creation of standard and guidelines for how to create "clean" packages, package creation tools, and package verification tools (against a central package database). This is absolutely essential to improve the quality of released packages and get a stable desktop platform.

For the future, with Office 2000 [Office2000] as well as NT 2000, Microsoft Software Installer [MSI, MSIT] (MSI) is looming on the horizon together with new release of InstallShield [InstallS] as well as SMS 2.0. We have started working on how we can replace part of our in-house developed components and integrate them with MSI.

Notice that the ideas expressed with a state driven installation can be realized using any installation format as long as the state is saved on the target system and in the user registry and/or home drive. There must also exist one-to-one mapping between a package version and the stored state to enable us to uniquely determine which version was installed.

6 Acknowledgments

The work described is the work of a lot of people over the last four to five years at UBS, and an incomplete list: Andre Aeppli, Martin Hufschmid, Diedrich Klarmann, Peter Kurz, Nick Riordan, Martin Schaible, Jeffrey Tolmie. And especially thanks to those I have forgotten.

References

[SMS1.2] R. Anderson, J. Farhat et al, *Microsoft SMS 1.2 Administrator's Survival Guide*, Sams Publishing, (1997).

[InstallS] *InstallShield for Windows Installer: Overview, White Paper*, Version 1.0, December (1998).

[LDAP] T. Howes and M. Smith, *LDAP: Programming Directory-Enabled Applications with Lightweight Directory Access Protocol*, Macmillan Technical Pub., (1997).

[MSI] M. Kelly, *Gain Control of Application Setup with the New Windows Installer*, Microsoft Systems Journal Sept (1998) p. 15-27.

[MSIT] *Microsoft's Software Installation Technology. Part 1: Client-side Installation Service*, Directions on Microsoft (Research), March (1998).

[Office2000] *Microsoft Office 2000 Deployment and Maintenance, White Paper*, `http://www.microsoft.com/office`

[SysV] *System V Packaging Manual*, `http://docs.sun.com`

[Win5] *WinInstall version 5.1*

[GINA] *Winlogon User Interface*, Microsoft Win32 Software Development Kit for Microsoft Windows.

A A System V Style Packaging System for Windows NT

The installation mechanism described is build on top of a packaging system for Windows NT which have been implemented in house since the middle 90's.

The application format is an implementation of a "System V" [SysV] like packaging system as under Solaris/SunOS, with improvements to enable a smoother integration with both the Windows NT and the Netware computing environment. We used concepts (`prototype` file, `pkgmap` file, etc.) from the System V packaging and also integrated features (variables and variable expansion in the output files) from the WinInstall [Win5, InstallS] product. We have kept the name of the standard packaging tools: `pkgmk` for package creation, `pkgadd` for package installation, `pkgchk` for package verification, and `pkgrm` for package removal.

We will try to give an overview of our packaging system as used in the environment with Windows NT clients connected to f&p servers. The target is to describe enough of the packaging system to make the installation mechanisms clear, without going into the more esoteric details of the actual implementation.

A.1 Background

Why packaging? Why not simply use WinInstall [Win5], InstallShield [InstallS], or the format as provided by the Microsoft SMS Installer [SMS1.2]?

When the Windows NT 4.0 project started in 90's, the existing technology was not good enough, and did not support both Netware servers as well as NT servers as target for application installation or parts thereof. We must support application installation to both server platforms and we had already a working packaging system for Netware servers. We also have experience with the System V packaging tools which we have extended extensively.

As installation targets, we must support NT workstations, NT servers, and Netware servers with a single packaging format for all platforms. We have a mixed environment where NT workstations can have a f&p server which is a NT server or a Netware server. And for the applications installed on the workstations, there must be no difference if the associated f&p server is a NT server or a Netware server.

Further, we must also support different languages on the server (English, German, French and Italian); flexible directory structure (not all application are installed into `C:\Program Files`; co-existing of 16-bit and 32-bit applications; configuration of application (e.g. where is the SQL server); and clean removal of applications.

A.2 Shared, Workstation and User Contexts

We generally talk about three different parts of a package: the shared context, the workstation context, and the user context. Before we go into details about the different contexts, we need to mention that a package can be installed in three modes:

Local mode is when everything in the package is installed on the target machine.

Shared mode is when part of the package is installed on the f&p server associated with a workstation. The typically installed components are dictionaries, help files, clip arts, or shared database files. In this case a set of workstations "share" the common items on the server.

Exported mode is only valid for a NT server (a shared application server) where a package is "exported" to a set of a workstations. The shared context of the package is installed on the server, but read/referenced by the clients.

It is important to notice that the package itself can be created (and should) in such a way to support both local and shared installation modes. Often the installation mode is selected at *installation time* and not at package creation time.

The *shared context* of a package are those files which are part of the shared installation. This can only be files and no registry entries, since the files are made available to the client via a SMB share[5] on the associated f&p server.

The *workstation context* of a package are file components and registry components which are installed on the client itself. The registry changes are made to the machine hive (HKLM[6]), often adding configuration information for the application or adding definitions for a DLL. The files are typically installed on the machine either in the `%SystemRoot%`, `%System32%`, or under `%SystemDrive%` directory. Typically a shared DLL (mfc42.dll) goes into %System32% and netscape.exe into `%SystemDrive%\ie_appl\ie_4\netscape`

[5] Typically `ie_appl`

[6] Hive Key Local Machine

The *user context* of a package contains the registry changes to the user hive (HKCU[7]), possible configuration files into the user home drive, or changes to the user profile. An typical example is to add a shortcut to the program menu, pointing to the application installed in the workstation context of the same package.

A.3 Packaging Classes

As under System V packaging, we provides a set of standard classes to perform the usual manipulations needed to install an application. All but one class is external, that is implemented as an executable outside of `pkgadd`:

none used for directory creation, file copying and also to copy the `install.txt` and the `pkginfo.ini` into pkginfo directory (internally implemented class).

registry class to do changes to the registry (`REG_SZ`, `REG_DWORD`, `REG_MULTI_SZ` etc.).

iniclass for changes to `.ini` files.

execbat to invoke `CMD` script during package add or package removal.

pkgenv to change the machine or user environment.

pkgpath to add or remove components to the machine (system) path or user path.

shortcut to create shortcut to an executable in the user's start menu or in the default/all menu for the machine.

pkgassoc to add an association between a file type (`.htm`) and an executable (`netscape.exe`).

pkghosts to add an hostname entry to `etc\hosts`

pkgserv to add a service definition to the `etc\service`

regocx to register a "self registration" DLL.

resolve to replace packaging variables in the input file with variable values (variable expansion).

[7]Hive Key Current User

In System V packaging, we have the concept of classes which are used to determine which part of a package is going to be installed. We do not have classes to do dynamical installation configuration, instead it is possible to place conditionals in the `prototyp.txt` as for the C pre processor (`#if...` `#else...` `#endif`). The conditional allow the package creator to query different package variables to determine which files to install. A typically examples is to install the correct sound card drivers dependent on the type of the machine.

A.4 Packaging Variables

In System V packaging, the `pkginfo` file is used together with the `request` script to gather the input necessary for the correct installation of the package. The environment created by the output from the `request` script can then be used during the `preinstall` and `postinstall` scripts.

In our implementation, we do not support interactive prompting (à la `request` script) for the install time configuration. All information must either be present on the machine at installation time, it must be possible retrieve from a configuration file, or it is possible to compute the configuration information.

We have the `pkginfo.ini` file. The following is an example from the standard UBS Perl package:

```
[PKGINFO]
PackageName=UBSPerl
PackageVersion=1.4
Description=Perl-Win32
ProgramName=perl
ProgramVersion=5
ProgramVendor=GNU
VendorVersion=5.004p2
ProgramLanguage=UN
PackageCategory=1
PackageType=0
PackageArchitecture=W32
InstallType=FULL
TargetArchitecture=LAN,STD
DiskSpace=6729603
DiskUser=0
DiskShared=0
DiskWorkstation=6729603
Restart=No

[R-Dependencies]
RT_MFC>=4,UN

[I-Dependencies]
RT_SYSTEM=4,IE

[Status]
PSTAMP=MARTIN970814135523
```

The variables listed in the section 'PKGINFO' are available during the whole packaging installation process. Further, for each machine, we have a packaging variable configuration file, **pkgvars.cfg**, which contains the standard mapping between the defined official variable names and the locations. This file also define the standard menu name and structure, the standard protection code for shared, workstation and user files. As an example for a Netware server with international English release:

```
System32         = "%SystemRoot%\system32"
UserRoot         = "%UserDrive%"
PackageData      = "%UserRoot%\%PackageName%"
AllUsers         = "%SystemRoot%\Profiles\All Users"
UserDrive        = "H:"
PackageDir       = "%ApplRoot%\%PackageID%"
UserMenu         = "%UserProfile%\Start Menu"
StartupMenu      = "%ProgramMenu%\Startup"
ReleaseDir       = "IE_APPL"
ProgramMenu      = "%UserMenu%\Programs"
DeveloperMenu    = "%ProgramMenu%\Development Tools"
DocumentsMenu    = "%DeveloperMenu%\Documents"
OfficeMenu       = "%ProgramMenu%\Office Applications"
PersonalMenu     = "%ProgramMenu%\Personal"
ApplDrive        = "I:"
ApplRoot         = "\\%server%\SYS2\%ReleaseDir%"
```

By having different configuration files for NT servers and Netware servers, we hide the differences in the directory structure. We also define in the guidelines which variables can be used and in which context, e.g. in the shared context your are not allowed to install a DLL into **%System32%** directory. You can only install a DLL into this directory in workstation context.

The variables can be used in the **prototyp.txt** and in all the files which are input to the external classes. By using variables in the file input to **registry** class, we will have the correct path to the executable:

```
REGEDIT4

[HKEY_LCCAL_MACHINE\SCFTWARE\\Perl]
'PERL5LIE'='%SystemDrive#%\\UES_Tccls\\%FrcgramName#%\\lib;'

[HKEY_LCCAL_MACHINE\SCFTWARE\Classes\.pl]
@='Perl'

[HKEY_LCCAL_MACHINE\SCFTWARE\Classes\Perl\Shell\Cpen\Command]
@='perl.exe %1 %*'
```

So far, the packaging variables described only have allowed configuration based on the static settings stored on the target machines, using the configuration files and the installation target. We have extended the packaging system to allow installation time querying of variables values, so called *EUP Values*. The actual value of the variable can be a single string value or multiple values. All variables which begin with a standard prefix are queried at installation time by **pkgadd** via an well defined interface exported by a DLL[8].

A.5 pkgmk

As under System V, the main input to the **pkgmk** is the **prototyp.txt** file which describes the components as well as into which context the different components are to be installed:

```
[Shared]
d none "%PackageDir%" ? ? ?

[Workstation]
!search ".\install"
e execbat "preTask.cmd" ? ? ?
!search ".\system32"
e registry "wks.reg" ? ? ?
f none "%System32%\nsrt2432.acm" RO ? ?
c #if %EUP_MachineType%=SERVER
  f none "%System32%\rt32cmp.dll" %WksFileAttr% PD ?
c #else
  f none "%System32%\rt32dcmp.dll" %WksFileAttr% P ?
c #endif
e execbat "postTask.cmd" ? ? ?

[User]
!search ".\install"
e registry "user.reg" ? ?  D
e shortcut "shortcut.sct" ?  ?  ?
```

The **pkgmk** generates a spooled package in a spool area with a standard directory layout. Included is the file **pkgmap.dat** describing the contents of the package which corresponds to the System V **pkgmap**.

A.6 pkgadd

For the installation of a package, or rather a context of package, we use **pkgadd**, specify the full package name, the package spool area, the target machine, and the selected context. Before we install the workstation context we must have a shared context installed, either locally on workstation or the associated the f&p server. This also applies for the user context, which cannot be installed until the workstation context have been installed.

Under System V packaging, we have the **preinstall** and **postinstall** standard scripts which we can use to prepare and/or for the cleanup of the package installation. This will have to be implemented via **BATCH** scripts located in the **prototyp.txt** in the first or last position within in each contexts. For example, see the **postTask.cmd** and **preTask.cmd** in the example **prototyp.txt**.

Under System V, the installation of a package modifies the **/var/sadm/install/contents** to add the

[8]By exchanging DLLs, we can have different data sources and transport mechanisms.

newly installed components and their protections together with the owner of the components. Also, a reference count of the packages which have installed the same components are kept in the `contents` file.

Instead of having a central database (the `contents` file), we selected to have a distributed state pro package and pro context in the local file system on the target machine and in the user home drive/ registry (part of the user profile). Notice that the same format and contents is used for all contexts and all targets. When the package is installed, `pkgadd` creates a "package information" (pkginfo) directory in the proper location on the target (machine and context) using the package name provided as argument to `pkgadd`. For a shared installed package, we will have a pkginfo directory for the shared context on the server; a pkginfo directory for the workstation context on the NT workstation; and a pkginfo directory in the user home drive and registry. Typically, the different locations for the example package `ubsperl_5_un_1.4`:

- `\\zhsv13664\PkgInfo\ubsperl_5_un_1.4\` for the shared context installation on a NT server.
- `\\zhbdvf02\Sys2\PkgInfo\ubsperl_5_un_1.4\` for the shared context installation on a Netware server.
- `%SystemRoot%\Config\PkgInfo\ubsperl_5_un_1.4\Shared\` for the shared context installed on a workstation (local install).
- `%SystemRoot%\Config\PkgInfo\ubsperl_5_un_1.4\` for the workstation context[9].
- `%HOMEDRIVE%\Sys32\PkgInfo\ubsperl_5_un_1.4\` for user context of the package[10].

In the pkginfo directory, we have the `pkginfo.ini` file, which is the same as provided in the package spool, except the information about the installation have been appended (when was the installation done, status, and who did the installation).

Further, we find the `pkgvars.ini` file, which contains the actual packaging variables, and their values, used for the installation of the package. We also have the `pkgstat.dat` file which contains the list of components installed in actual installation order.

All files which are used as input to any of the classes are also kept (e.g. input to *registry* class, `path` class etc.). Notice that the input files in the original package is processed and all the package variables are replaced with their values before the files are feed to the classes. By keeping the input file to the classes, we will not need the package spool during the package removal. Second, we ensure that the same values for the packaging variables are used for the package removal as were used for package installation.

A.7 pkgrm

For the removal of a package, we have `pkgrm` and all the necessary information for removal is present on the machine in the pkginfo directory. The package contexts for a single package should be removed in the reverse installation order, e.g. for a *local* installation the workstation context should be removed before the shared context. Notice that we should not remove the shared context installed on a server until all associated workstation have removed their workstation context. User contexts, being the last installed context for a package, can be removed at any time.

The `preremove` and `postremove` supported by System V have to be replaced with the `PreTask.cmd` etc. Notice that the installed components within the installed package are removed in reverse installation order using the `pkgmap.dat` file stored in the pkginfo directory.

[9]The location is stored in the registry on the machine and set when the machine configured. No hard coded paths are stored in tools themselves

[10]The exact path is stored in user registry and is initialized the first time `pkgadd` is called.