Building Packages for Sixnet Wireless Devices (D-Series/R3000/SN-6000/RAM-6000)

Rules for Packages

- All package names must end in .zip
- snupdate will only work if the package file in the /tmp directory
- snupdate does not support password protection on user-created packages
- It is recommended that users create packages on a unit which makes it easier to ensure that files have the correct paths, permissions, etc.

Modes of snupdate operation

snupdate has three modes of operation, based on the contents of the zip file it is called upon.

Unscripted Installation Packages

This is the default snupdate behavior. Unless it detects special contents in the zip file, snupdate treats the package as a plain zip file and extracts the contents with their full paths to the root directory of the device, thus all files should have the full path name when the package is created. This default behavior provides a simple way to update devices and install new files and programs, but does not provide the flexibility of the scripted installation method.

Scripted Installation Packages

If snupdate finds a file named install.sh in the root directory of a package, snupdate will treat the it as a scripted installation package. This means that snupdate will only extract the install.sh file to *tmp* and then run /tmp/install.sh, passing it a single argument of the package file name. The install.sh script is expected to extract the rest of the contents of the package itself and perform any other operations that the package requires. This method allows maximum flexibility but adds some complexity that may not be necessary for simple packages. For example, a scripted installation would be useful for a package that needs to check the current status or configuration of a unit (e.g. the firmware version) and behave differently based on that information.

Sample install.sh

```
#!/bin/sh

# Check that /etc/version exists. This pretty much only true on JBM/SN devices
if [ ! -e "/etc/version" ]; then
  echo "Unit does not appear to be a JBM/Sixnet unit, exiting."
  exit 1
fi

# The only argument we are passed is the file name of the package
FILE_NAME=$1

# Make sure we can find that file before we continue
if [ ! -e "$FILE_NAME" ]; then
  echo "Cannot find package file $FILE_NAME"
  exit 2
fi
```

```
# Extract the rest of the contents of the package
# -d             The destination directory for the unpacked contents
# -o             Overwrite existing files without prompting (use this!)
# $FILE_NAME     The file name of the package to unzip
# -x install.sh  Exclude install.sh when extracting since that has already been unpacked
unzip -d / -o $FILE_NAME -x install.sh

# Now that the contents are extracted, we can do any other action the package needs to perform
perl /tmp/xmllib.gmuclient.pl
```

Configuration Only Packages (gatherconfigs)

If snupdate does not find an install.sh script, but it does find all three of: /home/httpd/jbmconfig/conf/config.xml, /etc/rc.d/rc.local, and /etc/hosts, it will treat the package as a configuration package created by a gatherconfigs. This provides a fairly simple method to save the configuration of the unit and to restore that config or transfer it to another unit.

Examples of Common Tasks in Scripted Installs

Check Firmware Version

```
TOTAL_VERSION=`cat /etc/version | LANG=C awk 'BEGIN { FS = " " }; { print $3 }'`
VERSION=`echo "$TOTAL_VERSION" | LANG=C awk 'BEGIN { FS = "-" }; { print $1 }'`
RC_VER=`echo "$TOTAL_VERSION" | LANG=C awk 'BEGIN { FS = "-rc" }; { print $2 }'`
VMAJOR=`echo "$VERSION" | LANG=C awk 'BEGIN { FS = "." } ; { print $1 }'`
VMINOR=`echo "$VERSION" | LANG=C awk 'BEGIN { FS = "." } ; { print $2 }'`

if [ $VMAJOR = 1 -a $VMINOR -lt 46 ]; then
        echo "got a match"
fi

echo "
TOTAL_VERSION=$TOTAL_VERSION
VERSION=$VERSION
RC_VER=$RC_VER
VMAJOR=$VMAJOR
VMINOR=$VMINOR
"
```

Checking and installing for different CPU platforms

> Sometimes checking just the version of the build is not enough. Since the 3.xx build can run on both the D-series (MPC8313?) and also BT/SN devices (ARM926?) we need to be able to test for a specific CPU prior to unpacking binary files.

```
CPU=
if [ `grep "MPC8313" /proc/cpuinfo -c 2> /dev/null` -gt 0 ]; then
   CPU="MPC8313"
fi

if [ `grep "ARM926" /proc/cpuinfo -c 2> /dev/null` -gt 0 ]; then
```

```
   CPU="ARM926"
fi

if [ -z "$CPU" ];then
   echo "ERROR: This package does not appear to be for this type of system"
   exit 1
fi
```

Obviously a test for both CPUs? is not desired when installing for either one or the other. But sometimes to save space, or mainly confusion and errors, one might want to deploy an all-in-one package for both, where both binaries are in the same zip file. This can become somewhat complicated, but is useful if you pay attention and do it correctly. This next example was created to install the compiled program "tip" on 3 different systems, i386, PPC and ARM. It includes the version of the binary in the scripts, and in the file names zip file. Here is the meat of the script :

```
INSTALL_PROG=0
grep -E 'Version 3\.0(3|4|5|6|7|8)' /etc/version  &>/dev/null
if [ $? = "0" ]; then
   INSTALL_PROG=1
fi
grep -E 'Version 2\.' /etc/version  &>/dev/null
if [ $? = "0" ]; then
   INSTALL_PROG=1
fi



CPU=
if [ `grep "MPC8313" /proc/cpuinfo -c 2> /dev/null` -gt 0 ]; then
   CPU="MPC8313"
fi
if [ `grep "ARM926" /proc/cpuinfo -c 2> /dev/null` -gt 0 ]; then
   CPU="ARM926"
fi
if [ `grep "486 DX" /proc/cpuinfo -c 2> /dev/null` -gt 0 ]; then
   CPU="I386"
fi

if [ -z "$CPU" ];then
   echo "ERROR: This package does not appear to be for this type of system"
   exit 1
fi

#unzip everything but the script
unzip -o $FILE_NAME -x install.sh -d /

PROG_INST_DIR="/bin"
PROG_REAL_NAME="tip"
PROG_VER="1.1.0g"
PROG_NAME="${PROG_REAL_NAME}_${PROG_VER}_$CPU"
TMP_PROG_NAME="/tmp/${PROG_REAL_NAME}_${PROG_VER}_$CPU"

if [ "$INSTALL_PROG" = "1" ]; then
        if [ -e "$TMP_PROG_NAME" ]; then
           echo "Installing $PROG_REAL_NAME v$PROG_VER"
           chmod +x $TMP_PROG_NAME
           cp -f $TMP_PROG_NAME $PROG_INST_DIR/$PROG_REAL_NAME
```

```
            rm -f $FILE_NAME
         else
            echo "ERROR: Missing file $PROG_NAME, aborting"
            rm -f /tmp/$PROG_REAL_NAME*
         fi
fi
rm -f /tmp/$PROG_REAL_NAME\_$PROG_VER*
```

And here is how the zip file contents are laid out :

```
.
|-- install.sh
`-- tmp
    |-- runtip.sh
    |-- tip_1.1.0g_ARM926
    |-- tip_1.1.0g_I386
    `-- tip_1.1.0g_MPC8313

1 directory, 6 files
```

As you can see, there are 3 compiled binaried in a special location in "/tmp" so when they are unpacked they do not just simply overwrite /bin/tip, but can be installed based on the Build version and the CPU type. Attached is the install file this example came from.