

Chapter 29 KIDS Programmer Tools: Advanced Build Techniques

The previous chapter introduced KIDS from the developer's perspective, describing the basics of how to create build entries and how to transport distributions. This chapter describes advanced build techniques that developers can use when creating builds. The following subjects are covered:

- Environment Check Routine
- PRE-TRANSPORTATION ROUTINE
- Pre- and Post-Install Routines: Special Features
- Obtaining Package Name and Version Information
- Edit a Build - Screen 4
- How to Ask Installation Questions
- Using Checkpoints (Pre- and Post-Install Routines)
- Required Builds
- Package File Link
- Track Package Nationally
- Alpha/Beta Tracking
- Callable Entry Point Summary

Environment Check Routine

KIDS, like DIFROM, lets you specify an environment check routine. Typically, the environment check routine looks at the installing system and determines whether it's appropriate to install the package, based on conditions on the installing site's current system or environment.

You are not required to specify an environment check in order for your package to be installed. If, however, you have some special checks that you want to make to decide whether it's appropriate to go ahead with the installation, the environment check routine is the place to do it.

KIDS lets you specify the name of the environment check routine in screen one of EDIT A BUILD.

Self-Contained Routine

The environment check routine itself must be a single, self-contained routine. The reason is that it is the only routine from your build that will be loaded on the installing site's system at the time it is executed by KIDS. Based on what you find out about the installing system during the environment check, you can tell KIDS to continue installing the package, abort installing the package, or abort installing all packages (transport globals) in the distribution.

Although output during the pre-install and post-install should be done with the MES^XPDUTL and BMES^XPDUTL entry points, during the environment check routine you should use direct reads and writes.

Environment Check is Run Twice

KIDS runs the environment check routine twice. It runs the environment check routine first when the installer loads the transport global from the distribution (with the Load a Distribution option).

KIDS runs the environment check a second time when the user runs the Install Package(s) option to install the packages in the loaded distribution.

The KIDS key variable XPDENV (see below) indicates which phase (load or install) the environment check is running in.

Key Variables during Environment Check

XPDNM The KIDS key variable XPDNM is available during the environment check, as well as during the pre- and post-install phases of a KIDS installation. XPDNM is set to the name of the transport global currently being installed. It is in the format of the .01 field of the package's BUILD file entry, which is package name, concatenated with a space, concatenated with version number.

XPDENV The KIDS key variable XPDENV is available during the environment check only. It can have the following values:

- 0 The environment check is being run by the KIDS Load a Distribution option.
- 1 The environment check is being run by the KIDS Install Package(s) option.

You can use XPDENV if, for example, there is a check that is valid to perform at install time, but not at load time.

DIFROM For the purpose of backward compatibility, the variable DIFROM is available during the environment check, as well as during the pre- and post-install phases of a KIDS installation. DIFROM is set to the version number of the incoming package.

Package Version vs. Installing Version

KIDS provides several functions that you can use during the environment check to compare version numbers of the current package at the site to the incoming transport global. See the Obtaining Package Name and Version Information section in this chapter for more on this subject.

Telling KIDS to Skip Installing or Delete a Routine

During the environment check, you can tell KIDS to skip installing any routine and also change a routine's installation status to delete at site.

For example, suppose you have one version of a routine for MSM sites, one version for DSM for OpenVMS sites, and one version for DataTree sites. Based on the type of system your environment check finds, you can use the \$\$RTNUP^XPDUTL function to tell KIDS which routines to skip installing.

Verifying Patch Installation

During the environment check, you can tell KIDS to verify that a particular patch has been installed on a system prior to the installation of your package.

For example, if your package is dependent on a particular patch being installed, you can use the `$$PATCH^XPDUTL` function to have KIDS alert the user that a required patch is not installed on their system.

Aborting Installations During the Environment Check

In the environment check you can decide whether an installation should continue or stop, or whether the installation of all transport globals in the distribution should be aborted.

When you abort the installation of a transport global by setting `XPDQUIT` or `XPDABORT`, KIDS outputs a message to the effect that a particular transport global in the installation is being aborted. You should also issue your own message when aborting an installation, however, to give the site some diagnostic information as to why you've chosen to abort the install.

The following table lists ways you can ask KIDS to continue or abort an installation, based on the conclusions of your environment check routine:

Desired Action, Based on Environment Check Conclusions	How to Tell KIDS to Take This Action (during Environment Check)
OK to install this transport global.	(Take no action)
Don't install this transport global and kill it from ^XTMP.	S XPDQUIT =1
Don't install this transport global but leave it in ^XTMP.	S XPDQUIT=2
Abort another transport global named pkg_name in distribution and kill it from ^XTMP.	S XPDQUIT(pkg_name)=1
Abort another transport global named pkg_name in distribution but leave it in ^XTMP.	S XPDQUIT(pkg_name)=2
Abort all transport globals in distribution and kill them from ^XTMP.	S XPDABORT=1
Abort all transport globals in distribution but leave them in ^XTMP.	S XPDABORT=2

NOTE: It is recommended you use XPDQUIT when you have a distribution that contains multiple builds and you only want to selectively install a portion of it. Use the XPDABORT to abort the entire installation of a distribution.

Controlling the Queueing of the Install Question

By default, KIDS allows the installer to run in the future. It does this by allowing the installer to enter "Q" at the device prompt. If the variable XPDNOQUE is set to 1, then the installer will see the following prompt and not be allowed to enter "Q":

```
Enter the Device you want to print the Install messages.
Enter a '^' to abort the install.

DEVICE: HOME//
```

Controlling the Disable Options/Protocols Question

By default, KIDS asks the following question during KIDS installations:

```
Want to DISABLE Scheduled Options, Options, and Protocols? YES//
```

You can control the way this question is asked by defining the array XPDDIQ("XPZ1") during the environment check. The environment check runs once during the installation and prompts the user if it should run during the load. Setting this array only has an effect during the installation. Therefore, you may want to define the array only when XPDENV=1. You can use this array as follows (each node is optional):

XPDDIQ("XPZ1")	Set to 0 to force answer to NO or set to 1 to force answer to YES. When XPDDIQ("XPZ1") is set, the site is not asked the question.
XPDDIQ("XPZ1","A")	Replace default question prompt with value of this node.
XPDDIQ("XPZ1","B")	Set this node to new default answer, in external form ("YES" or "NO").

Controlling the Move Routines to Other CPUs Question

By default, KIDS asks the following question during KIDS installations:

```
Want to MOVE routines to other CPUs? NO//
```

You can control the way this question is asked by defining the array XPDDIQ("XPZ2") during the environment check. The environment check runs twice (once during load and once during installation), but setting this array only has an effect during the installation. Therefore, you may want to define the array only when XPDENV=1. You can use this array as follows (each node is optional):

- XPDDIQ("XPZ2") Set to 0 to force answer to NO, or set to 1 to force answer to YES. When this node is set, the question will not be asked.
- XPDDIQ("XPZ2","A") Replace default question prompt with value of this node.
- XPDDIQ("XPZ2","B") Set node to new default answer in external form ("YES" or "NO").

■ Sample Environment Check Routine

```

ZZRON1      ;SFISC/RWF - CHECK TO SEE IF OK TO LOAD ; 8 Sep 94 10:39
            ;;8.0T13;KERNEL; ;Aug 01, 1994
            N Y
            I $$($D(DUZ)[0:1,$D(DUZ(0))[0:1,'DUZ:1,1:0] W !!,*7,">> DUZ and
DUZ(0) must be defined as an active user to initialize." S XPDQUIT=2
            I $D(^DD(200,0))[0,XPDMN'["VIRGIN INSTALL" W !!, "You need to
install the KERNEL - VIRGIN INSTALL 8.0 package, instead of this
package!!" G ABRT
            ;check for Toolkit 7.3
            I $$VERSION^XPDUTL("XT")<7.3 W !!, "You need Toolkit 7.3
installed!" G ABRT
            ;
            W !!, "I'm checking to see if it is OK to install KERNEL
v", $P($T(+2), ";", 3), " in this account.", !
            W !!, "Checking the %ZOSV routine" D GETENV^%ZOSV
            I $P(Y, "^", 4)="" W !!, "The %ZOSV routine isn't
current.", !, "Check the second line of the routine, or your routine
map table." S XPDQUIT=2
            ;must have Kernel 7.1
            S Y=$$VERSION^XPDUTL("XU") G:Y<7.1 OLD
            ;Test Access to % globals, only check during install
            D:$G(XPDENV) GBLOK
            I '$G(XPDQUIT) W !!, "Everything looks OK, Lets continue.", !
            Q
            ;
            OLD W !!,*7, "It looks like you currently have version ",Y," of
KERNEL installed."
            W !!,*7, "You must first install KERNEL v7.1 before this version
can be installed.", !
            ;abort install, delete transport global
            ABRT S XPDQUIT=1
            Q
            ;
            GBLOK ;Check to see if we have write access to needed globals.
            W !!, "Now to check protection on GLOBALS.", !, "If you get an
ERROR, you need to add Write access to that global.", !
            F Y="^%ZIS", "^%ZISL", "^%ZTER", "^%ZRTL", "^%ZUA" W !!, "Checking
",Y S @(_Y_=$G("_Y_"))
            Q

```


PRE-TRANSPORTATION ROUTINE field (#900)

The PRE-TRANSPORTATION ROUTINE field (#900) in the BUILD file (#9.6) contains a [TAG^]ROUTINE that is run during the transportation process for the Build. This allows developers to populate the transport global using the variable XPDGREF.

Developers can now put information in the KIDS Transport Global which can be used by the Pre- or Post-install routines. KIDS runs the [TAG^]ROUTINE in the field PRE-TRANSPORTATION ROUTINE during the transport process. This routine can use the variable XPDGREF to set nodes in the transport global. For example:

```
S @XPDGREF@("My Namespace",1)="Information I need during install"
```

During the install process, in the Pre- or Post-install routines, the developer can retrieve the data by using the same variable, XPDGREF. Since these nodes are part of the transport global, they are removed when the install is completed.

■ PRE-TRANSPORTATION ROUTINE Sample

```

                                Edit a Build                                PAGE 1 OF 4
Name: TEST 4.0                                TYPE: SINGLE PACKAGE
-----
                                Name: TEST 4.0
                                Date Distributed: OCT 9,1996
                                Description:
Environment Check Routine:
                                Pre-Install Routine:
                                Post-Install Routine:
Pre-Transportation Routine: TAG^ROUTINE
-----
COMMAND                                Press PF1H for help                                Insert

```

Pre- and Post-Install Routines: Special Features

KIDS, like DIFROM, lets you specify pre-install and post-install routines. Typically, the pre- and post-install routines are used to perform pre-install and post-install conversions. This section describes how to use pre- and post-install routines with KIDS installations.

Pre- and post-routines are optional; you are not required to specify them in order for your package to be installed. If, however, you have some special actions you want to take, either before or after your installation, the pre- and post-install routines are the places to do it.

KIDS lets you specify the names for pre- and post-install routines in screen one of EDIT A BUILD.

Two new functions: \$\$OPTDE^XPDUTL and \$\$PRODE^XPDUTL can be called during the install process to disable or enable an option or protocol.

Don't set up variables during the pre-install for use during the installation or the post-install, because these variables will be lost if the installation aborts midway through and then is restarted by the site using the restart option.

You can reference any routine exported in your build, since all routines with a SEND TO SITE action are installed by the time the pre- and post-install routines run.

Aborting an Installation during the Pre-Install Routine

You can abort an installation during the pre-install routine by setting the variable XPDABORT to 1 and quitting. Note: this is exactly as if the installing site hit <CTRL>C, in the sense that no cleanup is done; options are left disabled. KIDS prints one message to the effect that the install aborted in the pre-install program. If you abort an installation in this manner, you need to tell the site what to do to either re-start the installation or clean up the system from the state it was left in.

Setting a File's Package Revision Data Node (Post-Install)

A new Package Revision Data node can now be updated during the **post**-install. This node is located in ^DD(filenum,0,"VRRV"). It is defined by the developer who distributes the package and may contain patch or revision information regarding the file. \$\$GET1^DID can be used to retrieve the content of the node and PRD^DILFD is used to update the node. See the *VA FileMan Programmer Manual* for more information.

Key Variables during Pre- and Post-Install Routines

- XPDNM** The KIDS key variable XPDNM is available during the pre- and post-install (as well as environment check) phases of a KIDS installation. XPDNM is set to the name of the build currently being installed. It is in the format of the .01 field of the package's BUILD file entry, which is package name, concatenated with a space, concatenated with version number.
- DIFROM** For the purpose of backward compatibility, the variable DIFROM is available during the pre- and post-install (as well as environment check) phases of a KIDS installation. DIFROM is set to the version number of the incoming package.
- ZTQUEUED** If the variable ZTQUEUED is present, you know that you're running as a queued installation. If ZTQUEUED is not present, you know that the installer chose to run the installation directly instead of queueing it.

Be Sure to NEW the DIFROM Variable when Calling MailMan

You are free to use the MailMan API to send mail messages during pre- and post-install routines (provided MailMan exists on the target system). Make sure that you NEW the DIFROM variable before calling any of the MailMan entry points, however. MailMan entry points may terminate prematurely if the DIFROM variable is present because the DIFROM variable has a special meaning within MailMan.

Update the Status Bar During Pre- and Post-Install Routines

During the installation, if the device selected for output is a VT100-compatible (or higher) terminal, KIDS displays the installation output in a virtual window on the terminal. Below the virtual window, a progress bar graphically illustrates the percentage complete that the current part of the installation has reached (see the Installation Progress Sample in the "KIDS System Management: Installations" chapter of the KIDS documentation). KIDS resets the status bar prior to the Pre- and Post-install routines.

You can provide a similar status bar for users in the Pre- and Post Install by doing the following:

1. Set XPDIDTOT=total number of items
2. Do UPDATE^XPDID(current number of items). This moves the status bar.

For example, if you are converting 100 records and want to update the user every time you have completed 10% of the records you would do the following:

```
Set XPDIDTOT=100
F%=1:1:100 D CONVERT I'(%#10) D UPDATE^XPDID(%)
```

If you wish to display a status bar at various intervals throughout your Pre or Post-install routines, you should reset the status bar. To reset the status bar do the following:

```
Set XPDIDTOT=0
D UPDATE^XPDID(0)
```

Callable Entry Points

For all output during pre- and post-installs, use the `MES^XPDUTL` and `BMES^XPDUTL` entry points. These functions write output to both the `INSTALL` file and the output device.

- **BMES^XPDUTL: Output Message with Blank Line**

Usage `D BMES^XPDUTL(msg)`

Input `msg:` String to output.

Output `none`

Description

For use during KIDS installations. Use this function to output a string to the installation device. Message is also recorded in `INSTALL` file entry for the installation. Similar to `MES^XPDUTL`, except that it outputs a blank line before it outputs the message, and it doesn't take arrays.

- **MES^XPDUTL: Output a Message**

Usage `D MES^XPDUTL([.]msg)`

Input `msg:` Message to output, either in a variable, or passed by reference as an array of strings.

Output `none`

Description

For use during KIDS installations. Use this function to output a message to the installation device. Message is also recorded in `INSTALL` file entry for the installation.

Obtaining Package Name and Version Information

- **\$\$PKG^XPDUTL: Parse Package from Build Name**

Usage `S Y=$$PKG^XPDUTL(buildname)`

Input `buildname:` Name of build (.01 field of BUILD file).

Output `return` Package name.
 `value:`

Description

Use this function to parse the name of a package from a package's build name. You can obtain the name of the build KIDS is installing from the KIDS key variable XPDNM which is defined throughout a KIDS installation.

- **\$\$VER^XPDUTL: Parse Version from Build Name**

Usage `S Y=$$VER^XPDUTL(buildname)`

Input `buildname:` Name of build (.01 field of BUILD file).

Output `return` Version of build identified in `buildname`
 `value:` parameter; null if no match in the BUILD file.

Description

Use this function to parse the version of a package from a package's build name. You can obtain the name of the build KIDS is installing from the KIDS key variable XPDNM, which is defined throughout a KIDS installation.

Edit a Build - Screen 4

Screen four of the EDIT A BUILD option is where you can set up the install questions, any required builds, PACKAGE file links, and tracking package information for a build.

■ Screen 4 of Edit a Build Sample

```

                                Edit a Build                                PAGE 4 OF 4
Name: TEST 1.0                                TYPE: SINGLE PACKAGE
-----
                                Install Questions
                                Required Builds
                                Package File Link...: TEST
                                Track Package Nationally: NO
-----
COMMAND:                                Press <PF1>H for help                                Insert
```

How to Ask Installation Questions

You are not required to ask any installation questions in order for your package to be installed. If, however, you have some special actions that you can take in your pre-install and post-install processes, and these special actions depend on information you need to get from your installer, then you need a way to ask these questions.

Screen four of EDIT A BUILD option is where you can set up the install questions for a build.

To ask questions, you need to supply KIDS with the proper DIR input values for each question. Then, KIDS uses the DIR utility to ask installation questions when performing installations. The DIR input values you can supply for each question are:

DIR(0)	question format
DIR(A)	question prompt
DIR(A,#)	additional message before question prompt
DIR(B)	default answer
DIR(?)	simple help string
DIR(?,#)	additional simple help
DIR(??)	help frame

For information on the purpose of these variables, permissible values for them, and which are required versus which are optional, please refer to the *VA FileMan Programmer Manual*.

Question Subscripts

For each question you want to ask, the .01 field of the question (as stored by KIDS) is a subscript. The subscript must be in one of two forms:

Pre-Install Questions	PRExxx
Post-Install Questions	POSxxx

"xxx" in the subscript can be any string up to 27 characters in length. KIDS asks questions whose subscript starts with PRE during the pre-install and questions whose subscript starts with POS during the post-install.

The order in which questions are asked during either the pre- or post- installs is the same as the sorting order of the subscript itself. KIDS asks questions with the lowest sorting subscript first and proceeds to the highest sorting subscript.

M Code in Questions

Besides specifying the DIR input variables, you can specify a line of M code which is executed after the DIR input variables have been set up but prior to the DIR call. The purpose of this line of M code is so that you can modify the DIR parameters, if necessary, before ^DIR is actually called.

The M code must be standalone, however; it cannot depend on any routine in the package (other than the environment check routine) since no other exported routines besides the environment check routine will be loaded on the installing system.

Skipping Installation Questions

If you want to prevent a question from being asked, you should kill the DIR variable in the line of M code for that question (execute K DIR).

Accessing Questions and Answers

Once the questions have been asked, the results of the questions are available (during pre-install and post-install only) in the following locations:

(Pre-Install Questions)

```

XPDQUES(PRExxx)=internal form of answer
XPDQUES(PRExxx,"A")=prompt
XPDQUES(PRExxx,"B")=external form of answer

```

(Post-Install Questions)

```

XPDQUES(POSxxx)=internal form of answer
XPDQUES(POSxxx,"A")=prompt
XPDQUES(POSxxx,"B")=external form of answer

```

The results of the questions for the pre-install can only be accessed (in XPDQUES) during the pre-install, and the results of the questions for the post-install can only be accessed (in XPDQUES) during the post-install. At all other times, XPDQUES is undefined for pre- and post-install questions.

■ Pre-Install Question (Setting Up) Sample

```

                                Edit a Build                                PAGE 4 OF 4
                                Install Questions
Name: PRE1

DIR(0): YA^^

DIR(A): Do you want to run the pre-install conversion?
DIR(A,#):

DIR(B): YES

DIR(?): Answer YES to run the pre-install conversion, NO to skip it...
DIR(?,#):
DIR(??):

M Code:

COMMAND:                                Press <PF1>H for help                                Insert

```

■ Appearance of Question During Installation:

```

Do you want to run the pre-install conversion? YES// ?
Answer YES to run the pre-install conversion, NO to skip it...
Do you want to run the pre-install conversion? YES//

```

Where Questions Are Asked During Installations

KIDS asks the pre- and post-install questions when a site initiates an installation of the package. The order of the questions is:

1. KIDS runs environment check routine, if any.
2. KIDS asks pre-Install questions.
3. KIDS asks generic KIDS installation questions.
4. KIDS asks post-Install questions.
5. KIDS asks site to queue the installation or run it directly.

Using Checkpoints (Pre- and Post-Install Routines)

A new feature of KIDS allows the installing site to restart installations that have aborted. This means that your pre-install and post-install routines must be "restart-aware:" that is, they must be able to run correctly whether it's the first time they're executed or whether it is the nth time through.

KIDS maintains a set of internal checkpoints during an installation. For each phase of the installation (for example, completion of each package component), it uses a checkpoint to record whether that phase of the installation has completed yet. If an installation errors out, checkpointing allows the installation to be restarted, not from the very beginning, but instead only from the last completed checkpoint onward.

In your pre- and post-install routines, you can use your own checkpoints. If there's an error during the pre- or post-install, and you use checkpoints, when the sites restart the installation, it will resume from the last completed checkpoint rather than running through the entire pre- or post-install again.

Another advantage of using checkpoints is that you can record timing information for each phase of your pre- and post-install routines, which allows you to evaluate the efficiency of each phase you define.

There are two distinct types of checkpoints you can create during pre- and post-install routines: checkpoints with call backs and check points without call backs.

Checkpoints With Call Backs

The preferred method of using checkpoints is to use checkpoints with call backs. When you create a checkpoint with a call back, you give the checkpoint an entry point (the call back routine). That is all you have to do during your pre- or post-install routine: create a checkpoint with a call back. You do not have to execute the call back. At the completion of the pre- or post-install routine, KIDS manages the created checkpoints by calling, running, and completing the checkpoint and its call back routine.

The reason to let KIDS execute checkpoints (by creating checkpoints with call backs) is to ensure that the pre-install or post-install runs in the same way whether it is the first installation pass, or if the installation aborted and has been restarted. If the installation has restarted, KIDS skips any checkpoints in the pre-install or post-install that have completed, and only executes the call backs of checkpoints that have not yet completed (and completes them).

In this scenario (checkpoints with call back routines), your pre-install and post-install routine should consist only of calls to `$$NEWCP^XPDUTL` to create checkpoints (with call backs). Once you create all of the checkpoints for

each discrete pre- or post-install task you need to accomplish, your pre-install or post-install should quit.

Once your pre- or post-install routine finishes, KIDS executes each created checkpoint (that has a call back) in the order you created them. If it is the first time through, each checkpoint is executed. If the installation has been restarted, KIDS skips any completed checkpoints, and only executes checkpoints that have not completed.

A summary of the KIDS checkpoint functions that apply when using checkpoints **with** call backs is:

<code>\$\$NEWCP^XPDUTL</code>	Create checkpoint (use during pre- or post-install routine only.)
<code>\$\$CURCP^XPDUTL</code>	Retrieve current check point name (use during pre- or post-install routine). Useful when using the same tag^routine for multiple call backs; this is how you determine which call back you're in.
<code>\$\$PARCP^XPDUTL</code>	Retrieve checkpoint parameter (use within call back routine.)
<code>\$\$UPCP^XPDUTL</code>	Update checkpoint parameter (use within call back routine.)

Checkpoint Parameter Node

You can store how far you have progressed with a task you're performing in the call back by using a checkpoint parameter node. The `$$UPCP^XPDUTL` entry point updates the value of a checkpoint's parameter node; the `$$PARCP^XPDUTL` function retrieves the value of a checkpoint's parameter node.

Being able to update and retrieve a parameter within a checkpoint can be quite useful. For example, if you're converting each entry in a file, as you progress through the file you can update the checkpoint's parameter node with the internal entry number (ien) of each entry as you convert it. Then if the conversion errors out and has to be re-started, you can write your checkpoint call back in such a way that it always retrieves the last completed ien stored in the checkpoint's parameter node. Then, it can process entries in the file starting from the last completed ien, rather than the first entry in the file. This is one example of how you can save the site time and avoid re-processing.

■ Using Checkpoints with Call Backs: Combined Pre- and Post-Install Routine

The pre-install entry point in this example is PRE^ZZRON2; the post-install entry point is POST^ZZRON2.

```

ZZRON2    ;RON TEST 1.0 PRE AND POST INSTALL
          ;;1.0
          ;build checkpoints for PRE
PRE       N %
          S %=$$NEWCP^XPDUTL("ZZRON1","PRE1^ZZRON2","C-")
          Q
PRE1     ;check terminal type file
          N DA,UPDATE,NAME
          ;quit if answer NO to question 1
          Q:'XPDQUES("PRE1")
          S UPDATE=XPDQUES("PRE2")
          ;write message to user about task
          D BMES^XPDUTL("Checking Terminal Type File")
          ;get parameter value to initialize NAME
          S NAME=$$PARCP^XPDUTL("ZZRON1")
          F S NAME=$O(^%ZIS(2,"B",NAME)) Q:$E(NAME,1,2)'="C- " D
          .S DA=+$O(^%ZIS(2,"B",NAME,0))
          .I DA,$D(^%ZIS(2,DA,1)), $P(^1,U,5)]" D MES^XPDUTL(NAME_"
still    has data in field 5") S:UPDATE $P(^%ZIS(2,DA,1),U,5)="
          .;update parameter NAME
          .S %=$$UPCP^XPDUTL("ZZRON1",NAME)
          Q
          ;build checkpoints for POST
POST     N %
          S %=$$NEWCP^XPDUTL("ZZRON1","POST1^ZZRON2")
          S %=$$NEWCP^XPDUTL("ZZRON2")
          Q
POST1    ;check version multiple
          N DA,VER,%
          ;quit if answer NO to question 1
          Q:'XPDQUES("POST1")
          ;write message to user about task
          D BMES^XPDUTL("Checking Package File")
          ;get parameter value to initialize DA
          S DA=+$$PARCP^XPDUTL("ZZRON1")
          F S DA=$O(^DIC(9.4,DA)) Q:'DA D
          .S VER=+$$PARCP^XPDUTL("ZZRON2")
          .F S VER=$O(^DIC(9.4,DA,22,VER)) Q:'VER D
          ..;here is where we could do something
          ..;update parameter VER
          ..S %=$$UPCP^XPDUTL("ZZRON2",VER)
          .;update parameter DA
          .S %=$$UPCP^XPDUTL("ZZRON1",DA),%=$$UPCP^XPDUTL("ZZRON2",VER)
          Q

```

Checkpoints Without Call Backs (Data Storage)

KIDS ignores checkpoints that don't have call back routines specified. The ability to create checkpoints without a call back routine is provided mainly as a facility for programmers to store information during the pre- or post-install routine. The parameter node of the checkpoint serves as the data storage mechanism. It is not safe to store important information in local variables during pre- or post-install routines: because installations can now be re-

started in the middle, variables defined prior to the restart may no longer be defined after a restart.

An alternative use lets you expand the scope of checkpoints without call backs beyond simply storing data. If you want to manage your own checkpoints instead of letting KIDS manage them, you can create checkpoints without call backs, but use them to divide your pre- and post-install routine into phases. Rather than having KIDS execute and complete them (as happens when the checkpoint has a call back routine), you would then be responsible for executing and completing the checkpoints. In this style of coding a pre- or a post-install routine, you would:

1. Check if each checkpoint exists (`$$VERCP^XPDUTL`); if it doesn't exist, create it (`$$NEWCP^XPDUTL`).
2. Retrieve the current checkpoint parameter as the starting point if you want to (`$$PARCP^XPDUTL`); do the work for the checkpoint; update the parameter node if you want to (`$$UPCP^XPDUTL`).
3. Complete the checkpoint when the work is finished (`$$COMCP^XPDUTL`).
4. Proceed to the next checkpoint.

You have to do more work this way than if you let KIDS manage the checkpoints (by creating the checkpoints **with** call back routines).

A summary of the KIDS checkpoint functions that apply when using checkpoints **without** call backs is:

<code>\$\$NEWCP^XPDUTL</code>	Create checkpoint (use during pre- or post-install routine.)
<code>\$\$PARCP^XPDUTL</code>	Retrieve checkpoint parameter (use during pre- or post-install routine.)
<code>\$\$UPCP^XPDUTL</code>	Update checkpoint parameter (use during pre- or post-install routine.)
<code>\$\$VERCP^XPDUTL</code>	Verify if checkpoint exists and if it has completed (use during pre- or post-install routine.)
<code>\$\$COMCP^XPDUTL</code>	Complete checkpoint (use during pre- or post-install routine.)

- **\$\$NEWCP^XPDUTL: Create Checkpoint**

Usage	S Y=\$\$NEWCP^XPDUTL(name, [callback], [par_value])	
Input	name:	Checkpoint name.
	callback:	[optional] Call back (^routine or tag^routine reference).
	par_value:	[optional] Value to set checkpoint parameter to.
Output	return value:	Internal entry number of created checkpoint if created or if checkpoint already exists, or 0 if error occurred while creating checkpoint.

Description

For use during KIDS installations. Use this function to create a checkpoint, in pre- or post-install routines. The checkpoint is stored in the INSTALL file.

Pre-and post-install checkpoints are stored separately, so you can use the same name for a pre- and post-install checkpoint if you wish. Checkpoints created with this function from the pre-install routine are pre-install checkpoints; checkpoints created during the post-install routine are post-install checkpoints.

You can use \$\$NEWCP^XPDUTL to create a checkpoint with or without a call back. You can also store a value for the parameter node, if you wish.

Checkpoints created with call backs have that call back automatically executed by KIDS during the appropriate phase of the installation. If the checkpoint is created during the pre-install routine, KIDS executes the call back as soon as the pre-install routine completes. If the call back is created during the post-install, KIDS executes the call back as soon as the post-install routine completes. If multiple checkpoints are created during the pre- or post-install routine, KIDS executes the call backs (and completes the checkpoints) in the order the corresponding checkpoints were created.

Checkpoints created without a call back cannot be executed by KIDS; instead, they provide a way for developers to store and retrieve information during the pre-install and post-install phases. Rather than storing information in a local or global variable, you can store information in a checkpoint parameter node and retrieve it (even if an installation is re-started).

If the checkpoint you are trying to create already exists, the original parameter and call back will not be overwritten.

• **\$\$PARCP^XPDUTL: Get Checkpoint Parameter**

Usage	<code>S Y=\$\$PARCP^XPDUTL(name[,pre])</code>	
Input	name:	Checkpoint name.
	pre:	[optional] To retrieve a parameter from a pre-install checkpoint while in the post-install, set this parameter to "PRE".
Output	return value:	Current parameter node for checkpoint named in name parameter.

Description

For use during KIDS installations. Retrieves the current value of a checkpoint's stored parameter. The parameter is stored in the INSTALL file.

Use this entry point for checkpoints both with and without call backs.

Use the optional second parameter to retrieve a pre-install checkpoint's parameter during a post-install.

• **\$\$UPCP^XPDUTL: Update Checkpoint**

Usage	<code>S Y=\$\$UPCP^XPDUTL(name[,par_value])</code>	
Input	name:	Checkpoint name.
	par_value:	[optional] Value to set checkpoint parameter to.
Output	return value:	Internal entry number of updated checkpoint if successful or 0 if error updating checkpoint.

Description

For use during KIDS installations. Use this function to update the parameter node of an existing checkpoint, in pre- or post-install routines. The parameter node is stored in the INSTALL file.

Use this entry point for checkpoints both with and without call backs.

During the pre-install, you can only update pre-install checkpoints; during the post-install, you can only update post-install checkpoints.

- **\$\$VERCP^XPDUTL: Verify Checkpoint**

Usage SY=\$\$VERCP^XPDUTL(name)

Input name: Checkpoint name.

Output return 1 if checkpoint has completed, 0 if checkpoint
 value: has not completed but exists, -1 if checkpoint
 doesn't exist.

Description

For use during KIDS installations. Use this function to check whether a given checkpoint exists and, if it exists, whether it has completed or not.

Use this entry point only for checkpoints with no call back.

During the pre-install, you can only verify pre-install checkpoints; during the post-install, you can only verify post-install checkpoints.

Required Builds

In the fourth screen of the EDIT A BUILD option, you can use the Required Builds multiple to enter other builds (i.e., packages, or patches) that either warn the installer when they are missing or requires that they be installed before this build is installed. Make an entry in the BUILD file for those packages or patches not installed using KIDS. Include the name and version number in the BUILD file entry. For the action types available, see the Required Builds Installation Actions table below.

At the installing site, KIDS checks the PACKAGE file, VERSION multiple, and PATCH APPLICATION HISTORY multiple to verify that the required build has been installed at that site.

■ Required Builds Sample

```

                                Edit a Build                                PAGE 4 OF 4
Name: TEST 1.0                                TYPE: SINGLE PACKAGE
-----
                                Install Questions

                                Required Builds

TEST 1.1                                Don't install, remove global

                                Package File Link...: TEST
Track Package Nationally: NO
-----
COMMAND:                                Press <PF1>H for help    Insert

```


■ **Required Builds Installation Actions**

Installation Action	Description
WARNING ONLY	Warns the installer the listed package/patch is missing at the site but allows the installation to continue. (Displays a **WARNING** to the installer.)
DON'T INSTALL, LEAVE GLOBAL	If the listed package/patch is missing, this action prevents sites from continuing the installation. It does <i>not</i> unload the Transport Global. This allows sites to install the missing item and continue with the installation without having to reload the Transport Global.
DON'T INSTALL, REMOVE GLOBAL	If the listed package/patch is missing, this action prevents sites from continuing the installation. It also <i>unloads</i> the Transport Global.

PACKAGE FILE Link

In the fourth screen of the EDIT A BUILD option, you can link your build to an entry in the national PACKAGE file. Use this link if you want to update the site's PACKAGE file when the package you're creating is installed or if you want to use Kernel's Alpha/Beta Testing module. You can only link to a PACKAGE file entry that is the same name (minus the version number) as the build you're creating.

If you specify a PACKAGE file entry in the PACKAGE FILE LINK field, and the installing site does not have a matching entry in their PACKAGE file, KIDS creates a new entry in the installing site's PACKAGE file.

KIDS checks for duplicate version numbers and patch names when updating the PACKAGE file. When you link to an entry in the PACKAGE file, your installation automatically updates the VERSION multiple in the installing site's corresponding PACKAGE file entry. KIDS makes a new entry in the VERSION multiple for the version of the package you are installing. KIDS fills in the following fields in the new VERSION entry:

- VERSION
- DATE DISTRIBUTED
- DATE INSTALLED AT THIS SITE
- INSTALLED BY
- DESCRIPTION OF ENHANCEMENTS
- PATCH APPLICATION HISTORY
 - PATCH APPLICATION HISTORY
 - DATE APPLIED
 - APPLIED BY
 - DESCRIPTION

KIDS saves patch names along with their sequence numbers in the PATCH APPLICATION HISTORY multiple (this functionality was added in patch XU*8*30). The Patch Application History sample shows a list of patch names with and without sequence numbers. Those patches without sequence numbers were entered prior to patch XU*8*30, since no sequence numbers are evident.

In addition, you can choose to update the following fields at the top level of the National Package file:

- | | |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| PRIMARY HELP FRAME | Select the primary help frame for the package. |
| AFFECTS RECORD MERGE | (multiple) Select files that, if merged, affect this package. |
| ALPHA/BETA TESTING | YES means that this package is currently in alpha or beta test and that you want to track option usage and errors relating to this package at the sites.

NO means that you want to discontinue tracking of alpha or beta testing at the sites. |

Beyond these fields, KIDS does not support maintaining any other information in the PACKAGE file.

■ Patch Application History Sample

```
Select PATCH APPLICATION HISTORY: 48// ?
Answer with PATCH APPLICATION HISTORY
Choose from:
 27
 39
 41
 42
 48
 45 SEQ #41
 46 SEQ #42
 47 SEQ #43

    You may enter a new PATCH APPLICATION HISTORY, if you wish
    Answer must be 8-15 characters in length.

Select PATCH APPLICATION HISTORY: 48// <RET>
PATCH APPLICATION HISTORY: 48// <RET>
DATE APPLIED: SEP 20,1996// <RET>
APPLIED BY: DOE,JANE// <RET>
DESCRIPTION:
1>This contains fixes related to output fixes for the PCMM
software
2>(distributed as SD*5.3*41).
3>
4>Both SD*5.3*41 and SD*5.3*45 must be installed prior to loading
this
5>patch.
```


Track Package Nationally

The fourth screen of the EDIT A BUILD option also lets you choose whether to send a message to the National Package File on FORUM, each time the package is installed at a site. If you enter YES in the TRACK PACKAGE NATIONALLY field, KIDS sends a message to FORUM when a site installs the package, provided the following conditions are met:

- The PACKAGE FILE LINK field in the build entry points to an entry in the PACKAGE file.
- The package is installed at a site that is a primary VA domain.
- The package is installed in a production UCI.

Answering NO to TRACK PACKAGE NATIONALLY (or leaving it blank) means that KIDS does not send a message to FORUM.

Alpha/Beta Tracking

Alpha/Beta tracking provides the following services to developers:

- Notification when a new package version is installed.
- Periodic option usage reports.
- Periodic listings of errors in the package's namespace.

This section describes how to set up alpha/beta tracking for packages you export.

Setting up Alpha/Beta Tracking in the Build Entry

The first step to setting up alpha/beta tracking occurs when you are creating your build entry. In the PACKAGE FILE LINK section of the build entry, you can turn on alpha/beta tracking by answering YES to Alpha/Beta Testing. KIDS places you in a form that lets you edit the following alpha/beta testing options:

Installation Message	Answering YES means that an installation message will be sent when this package is installed at a site. The message will be sent to the mail group specified in the ADDRESS FOR USAGE REPORTING field.
Address for Usage Reporting	Should be set to the MailMan address of a mail group at the developer's domain. This MailMan address is where installation and option usage messages are sent by the Alpha/Beta Tracking module. Also, the domain specified in the address is where server requests are sent from the sites to report errors.
Select Package Namespace or Prefix	This field is where you identify the alpha/beta package namespaces to track.

There is additional setup required to enable alpha/beta tracking, however. The remaining tasks are:

- The server option must be set up correctly at the development domain.
- Errors Logged in the Alpha/Beta Test (Queued) option should be scheduled to run at sites to gather errors and report these to the development server.

- Send Alpha/Beta Usage to Developers option should be scheduled at the sites to send mail messages containing option usage.

The Site Option to Report Errors

```
ZTMQUEUABLE OPTIONS                                [ZTMQUEUABLE OPTIONS]
Errors Logged in Alpha/Beta Test (QUEUED)         [XQAB ERROR LOG XMIT]
```

The Errors Logged in Alpha/Beta Test (QUEUED) option identifies any errors associated with an application package which is in either alpha or beta test. You should instruct your test sites to schedule it as a task to run daily, after midnight.

The identified errors are combined in a mail message which includes the type of error, routine involved, date (usually the previous day), the option which was being used at the time of the error, and the number of times the error was logged. The volume and UCI are included so that stations with error logs being maintained on different CPUs can run the task on each different system.

Collecting Error Reports (Developer's Domain)

In order to track errors at test sites, make sure that the server option [XQAB ERROR LOG SERVER] resides at your development site (which should be the domain specified in the ADDRESS FOR USAGE REPORTING field in the build entry).

This option processes server requests from the test sites, from the [XQAB ERROR LOG XMIT] option. The server stores the data from the requests into the XQAB ERRORS LOGGED file.

You can use the Print Alpha/Beta Errors (Date/Site/Num/Rou/Err) option to print out the collected errors.

```
Operations Management ...                          [XUSITEMGR]
Alpha/Beta Test Option Usage Menu ...             [XQAB MENU]
Print Alpha/Beta Errors (Date/Site/Num/Rou/Err)
                                                    [XQAB ERR DATE/SITE/NUM/ROU/ERR]
```


Send Alpha/Beta Usage to Developers Option

To receive option usage reports, you should instruct the sites to schedule this option to run at whatever frequency you want to receive option usage reports. The option can also be run manually by the sites to send option usage information. Mail messages are sent to the mail group specified in the build entry for Address for Usage Reporting. Make sure that this mail group exists at your development domain!

Turning Off Alpha/Beta Tracking

Alpha/Beta Tracking, once initiated for a package, should be turned off when the final version of the package is released. You can turn off alpha/beta tracking by answering NO to the ALPHA/BETA TESTING field in the build. When the sites install the build, tracking is shut off.

To manually shut down tracking at an individual site, they can use the Enter/Edit Kernel Site Parameters option to remove the desired entries from the ALPHA/BETA TEST PACKAGE multiple in the KERNEL SYSTEM PARAMETERS file.

KIDS Callable Entry Point Summary

■ KIDS Environment Check Functions

Entry Point	Description
<code>\$\$RTNUP^XPDUTL(routine,action)</code>	Update routine installation action (install, delete, or skip).
<code>\$\$PATCH^XPDUTL(patch)</code>	Verify if a patch has been loaded.

■ KIDS Installation Output Functions

Entry Point	Description
<code>BMES^XPDUTL(msg)</code>	Output a string during install, preceded by blank line.
<code>MES^XPDUTL([.]msg)</code>	Output a string during install.

■ KIDS Installation Protocol and Option Functions

Entry Point	Description
<code>\$\$PRODE^XPDUTL(name,action)</code>	Disable/Enable a protocol.
<code>\$\$OPTDE^XPDUTL(name,action)</code>	Disable/Enable an option.

■ KIDS Package Name and Version Functions

Entry Point	Description
<code>\$\$PKG^XPDUTL(buildname)</code>	Parse package name from build name.
<code>\$\$VER^XPDUTL(buildname)</code>	Parse package version from build name.
<code>\$\$VERSION^XPDUTL(package_id)</code>	Return site's currently installed package version from PACKAGE file.

■ KIDS Check Point Functions

Entry Point	Description
<code>\$\$COMCP^XPDUTL(name)</code>	Complete a checkpoint.
<code>\$\$CURCP^XPDUTL(format)</code>	Get current checkpoint name or internal entry number.
<code>\$\$NEWCP^XPDUTL(name, [callback], [par_value])</code>	Create a checkpoint.
<code>\$\$PARCP^XPDUTL(name[,pre])</code>	Get checkpoint parameter node.
<code>\$\$UPCP^XPDUTL(name[,par_value])</code>	Update a checkpoint.
<code>\$\$VERCP^XPDUTL(name)</code>	Verify a checkpoint.