EPICS Application Source/Release Control

Bob Zieman, Janet Anderson and Marty Kraimer
Argonne National Laboratory, Advanced Photon Source
appSR R1.3, May 1, 1995
EPICS Release 3.12

Table of Contents

1. Introduction

Overview

This manual describes a set of Application Source/Release Control tools (appSR) that can be used to develop software for EPICS based control systems.

Overview to EPICS R3.12 Version

The Application Source/Release Control System (appSR) has been unbundled from base EPICS and is now available as an EPICS extension. Due to this unbundling, two new directories must be added to a user's path (see section "Environment" on page 3 for more information) and a new command `getapp` must be issued after the `getrel` command to get a specific version of appSR (see section "Creating The Initial Application System Area" on page 7 for more information).

It is now required that GNU `make` version 3.71 or later be used for makes instead of SUN `make`. Users should now type `gmake` instead of `make`.

A new method of adding application specific source files is now supported. This method allows the user to create and use simple `Makefiles` in an application source directory (see section "Adding Application Specific Source Files" on page 13 for more information). `Makefile` templates are provided. `Imakefiles` and the `Buildit` command are no longer necessary. Targets built from these `Makefiles` will reside in target-architecture specific subdirectories of this source directory. New makefile targets, `clean`, `depends` and `build`, have been added to support the Makefile method. For upward compatibility, `Imakefiles` and the `Buildit` command are still supported but their use is not encouraged.

There are two changes to the Imakefile method. Depends are no longer available for Unix architectures and the "`gmake buildMakefiles`" command now executes "`gmake clean`" before executing "`gmake`" after the Makefiles are rebuilt.

Features

The Application Source/Release Control System (appSR) provides the following features:

Multiple Applications: The entire system is composed of an arbitrary number of applications.

Source/Release Control: All files created or modified by the application developers can be put under `sccs`, which is a Unix source/release control utility.

Multiple Developers: It allows a number of application developers to work separately during the development phase but combine their applications for system testing and for a production system.

Makefiles: Makefiles are provided to automatically rebuild various application components. For C and state notation programs Makefile templates are provided. For compatibility Imakefiles are supported.

Definitions

Application System Area: The set of directories and files managed by the tools described in this document. Everything is stored in one directory tree. The top level directory contains information common to all applications in this area as well as a subtree for each application.

Application: A subtree under the application system area that contains all the files for a single application.

Application Shadow Area: A set of directories and soft links to an application system area for use by the application developer. It appears to the user just like a copy of the system area. It is used for individual development and testing of application changes/fixes.

Application Production Area: A copy of a working application system area for use by operations.

Classes of Users

Application System Manager: The Application System Manager is responsible for the Application System area.

Application Developer: Anyone who tests, modifies, or extends an application's software. If multiple developers are working on the same system each should develop in a private shadow area.

Application Production Manager: A person responsible for production application software.

Document Conventions and Information

The following conventions and/or representations apply to the remainder of this document.

- **<appSR>** Represents the full path name of an appSR release
- **<epics>** Represents the full path name of an EPICS release
- **<top>** Represents the root node of an application system area. It is the directory from which we can access EPICS components.
- **<shadow>** Represents the top node of an application shadow node. "shadow node" and "shadow area" are synonymous.
- **<archV>** Represents the vxWorks target architecture. Currently `hkv2f` for the 68020 and `mv167` for the 68040.
- **<archU>** Represents the Unix architecture. Currently sun4.
- ***%*** Indicates a prompt for user input or activity.
- **<EDIT>** Means: edit the file according to the `sccs` rules. Refer to the `sccs` procedures section "Source/Release Control Commands" on page 3. Note: **<EDIT>** includes doing a `delget` if in the system area.

Environment

In order to use the Application Source/Release tools and executables you need to have the following item in your path:

```
./appSR/bin/<archU>  $path
```

Ask your EPICS site manager for the name of a file which you can source from your .cshrc file to satisfy the above appSR path requirements as well as all the EPICS Unix environment and path requirements.

At APS an EPICS user needs only the following statement in his/her .cshrc file placed after any set path statements.

```
source /usr/local/etc/Cshrc.aps
```

Source/Release Control Commands

The Unix `sccs` utility is used to put all user editable files under source/release control. The Unix documentation should be consulted for a complete description of `sccs`. This section gives a brief description of the commands normally used by application developers. Wherever `<filename>` is shown a list of filenames is allowed.

create

```
%sccs create <filename>
```

This command places a file under `sccs` control for the first time. After the file is placed under `sccs` control a read only copy is created, i.e. an `sccs edit` command must be issued before the file can be modified.

This command also creates a backup copy of the original file. It is the original file with a comma prepended to the name. It is a good idea to remove this file, now.

edit

```
%sccs edit <filename>
```

This command checks out a file so that it can be modified. If a file is checked out in an application shadow area, other developers will not see any modifications until an `sccs delta` is executed.

If a developer checks out a file in a shadow area it is actually checked out from the system area, i.e. no other developer can try to modify the same file. Other developers do not, however, see any changes made in the shadow area until the developer checks in the modified files.

unedit

```
%sccs unedit <filename>
```

This command causes the SCCS directory to revert to the state it was in before the last `sccs edit <filename>` command was issued.

delta

```
%sccs delta <filename>
```

This command checks in a modified file. This should only be done during an Application System area integration when everyone attached to an application system area is expected to see the changes. A new `sccs` version of the file is created. It is possible to retrieve previous versions.

It is also possible to issue an "`sccs delget`", which combines an `sccs delta` with an `sccs get`, and an "`sccs deledit`", which combines an `sccs delta` with an `sccs edit`.

get

```
%sccs get <filename>
```

This command retrieves a read only version of the file. It is useful in a shadow area when the user wants to make a temporary change to a file, e.g. for debugging purposes. In this case the user must change the file protections before it can be modified.

info

```
%sccs info
```

This command displays a list of all checked out files in the directory from which the command is issued.

Getting Started

The normal procedure for getting started is to:

1. **Create System Area**: This is done by the application system manager. See section "Procedures for Application System Area" on page 6.
2. **Get Application Specific ASCII Definition Files**: All application developers using this application system area must agree on a common set of ASCII definition files. See section "ASCII Definition Files" on page 12 for instructions.
3. **Run Makesdr**: After all ASCII files are installed run `makesdr` in `<top>`.
4. **Populate Each Application**: The developers for each application should install all files related to each application. This includes Unix sources, IOC sources, and databases. See section "Adding/Modifying Components" on page 10, which explains how to install each component. It is up to the application system manager and the application developers to decide if it is easier to do this in the system area or if each application developer should do his/her part in a shadow area.
5. **Prepare Each IOC For Booting**: Modify the startup files in each IOC directory.
6. **Perform The Normal Integration Steps**: Perform the steps given in section "Integration" on page 8.

At this time you should have a working application system area.

2. Application System Area Architecture

The root directory of the Application System Area and its contents is referred to as `<top>`. Appendix A shows the file structure stored under `<top>`.

<top>

User Editable Files

- **appList**: List of all applications
- **iocList**: List of all IOCs used by these applications

EPICS related links and files

The following are soft links to the EPICS base and appSR directories and files.

- **base**@: EPICS release base directory
- **appSR**@: appSR release directory

The following are soft links to EPICS directories or files.

- **ascii**@: ASCII definition files
- **base**@: EPICS release base directory
- **config**@: Directory containing files needed by source/release tools
- **epicsH**@: Include files
- **include**@: Include files (same as epicsH)
- **makefile**@: Top-level makefile
- **target<archV>**@: Directory containing EPICS vxWorks executables
- **vw**@: Location of vxWorks components
- **vxWorks<archV>**@: vxWorks boot image

The following file identifies EPICS releases.

- **.current_rel_hist**: History of all `getrel` commands for this system

The following files contain the record and device support definitions. The default files are initially links to EPICS and `sdrH` does not exist. `makesdr` creates these if they are missing or out-of-date.

- **default.dctsdr**: Record definitions
- **default.sdrSum**: Checksum file for record definitions
- **sdrH/rec/**: The include files for each record type.

<top>/cat_ascii

This is the place to store the application specific ASCII definition files to be added to the end of the existing EPICS definition files.

<top>/ replace_ascii

This directory contains the application specific ASCII definition files that are not part of EPICS and also files that replace the existing EPICS supplied files. It is also the place to store C include files containing definitions used in ASCII definition files.

<top>/ioc/ <iocName>/

These directories (one for each IOC) contain soft links to the EPICS components needed to boot an IOC. Each also contains a `st.cmd<archV>` file which must be customized for the particular IOC. Any modules to be loaded into the IOC must be referenced from the `st.cmd<archV>` file in a relative fashion. If other IOC specific files are needed, this is the place to put them. All user created files should be placed under `sccs` control.

<top>/ <app>App/

Root node of an application.

- **<top>/<app>App/src/**: This directory contains files that are meant to be edited by the application developer. This includes the following:
  - C source
  - Include files
  - State sequence programs
  - `Makefile`, `Makefile.Unix`, and `Makefile.Vx` and/or `ImakefileVx` and `ImakefileUnix`

  This is the directory for building both application specific IOC and Unix components from the `Makefile`. The `Makefile` will use the `Makefile.Unix` and `Makefile.Vx` files to build the components.
- **<top>/<app>App/src/O.<archV>/**: Directory created by `<src>/Makefile` for application specific IOC components. It contains a link to `<src>/Makefile.Vx` so that `gmake` can be executed.
- **<top>/<app>App/src/O.<archU>/**: Directory created by `<src>/Makefile` for application specific Unix components. It contains a link to `<src>/Makefile.Unix` so that `gmake` can be executed.
- **<top>/<app>App/archList**: This file does not exist in new application system areas. if it exists, it contains entries for each target architecture directory to be built (ex: `sun4`, `mv167`, `hkv2f`) using Imakefiles. You may remove the entries you do not want built from the Imakefiles.
- **<top>/<app>App/<archV>/**: This directory does not exist in new application system areas. If it exists, it is the directory for building application specific IOC components using Imakefiles. It contains a link to `<src>/ImakefileVx` so that `Buildit` and `gmake` can be executed.
- **<top>/<app>App/<archU>/**: This directory does not exist in new application system areas. If it exists, it is the directory for building application specific Unix components using Imakefiles. It contains a link to `<src>/ImakefileUnix` so that `Buildit` and `gmake` can be executed.
- **<top>/<app>App/<db>Db/:** Each application can have an arbitrary set of database directories and each database directory can contain an arbitrary number of IOC databases.
- **<top>/<app>App/op/**: This directory initially contains the following subdirectories:
  - adl/
  - alh/
  - ar/
    - arChan/
    - arReq/
    - arSet/
  - burt/
  - doc/
  - km/

  These directories contain files, for the specified tools, that the application developer wants to place under `sccs` control. The user may also define other directories and contents in this area.

3. Procedures for Application System Area

This section describes procedures that can only be executed in the system area not in a shadow area. In general any procedure that creates a new directory must be executed in the system area. This includes:

- New application directories
- New IOC directories
- New database directories within an application

It is up to the application developer to notify the application system manager of any new structures to be introduced.

Any **<EDIT>** commands shown in this chapter could also be executed in a shadow area but it is often easier to perform them in the system area.

Creating The Initial Application System Area

Setup the <top> Directory.

```
%mkdir <top>
%cd <top>
%<epics>/base/tools/getrel   <epics>
```

The `getrel` command to be executed must be executed from the same EPICS release that you will be using.

```
%<appSR>/bin/<archU>/getapp <appSR>
```

The `getapp` command to be executed must be executed from the same appSR release that you will be using.

```
%apCreateTop
```

This command creates any missing <top> directory components of an application system area.

```
%apFixLinks
```

Creating Application Specific Directories

Creating New Applications

In order to create the initial applications or add new applications execute the following commands:

```
%cd <top>
% <EDIT> appList
```

If `appList` is not out for edit issue the command "`sccs edit appList`**".** Use your favorite editor to add one or more application names to the file. Example: `appName1` or `appName2`

```
%apCreateApp
```

If you are using the Imakefile method of building IOC and Unix components execute the following two commands and then use your favorite editor to remove unwanted target architecture names from the archList file.

```
%cd <top>/<app>App
% <EDIT> archList
```

Creating New Database Directories

```
%cd <top>/<app>App/
%apCreateDbDir
```

This tool is interactive. You are prompted for each database name, followed by a prompt for the database editing tool (GDCT or DCT) to be used in that directory.

Note that only one database editor (either GDCT or DCT) may be used in a given directory.

Creating IOC Directories

```
%cd <top>
%<EDIT>  iocList
```

Use your favorite editor to add the new IOC names to the file. Example: `iocName1` or `iocName2`

```
%apCreateIocName
```

For each name in `iocList`, this tool creates a `<top>/ioc/<iocName>` directory and populates it with “default” templates and links. Each new directory contains an IOC startup file. The user should modify each `st.cmd<archV>` file to include the application components and/or instructions to load and run the application.

Integration

Integration means going to a new release of EPICS and/or going to a new release of appSR or updating the application system area to reflect changes made by application developers in their shadow nodes.

The primary steps are as follows:

1. The application system manager coordinates with the application developers.
2. The application developers should make sure that all DCT databases have an up to date short form report (`.rpt` files) which is not out for `sccs` edit. The application developer can issue the `apSccsInfo` command at the `<top>` level to get a list of files out for edit.
3. The application developers check in (`sccs delta`) from their shadow nodes any files they want to be part of the new system.
4. The application system manager issues the following commands:
   ```
   %cd <top>
   %<epics>/base/tools/getrel <epics>
   ```
   where <epics> is the full path name to the new version of EPICS.
   and/or
   ```
   %<appSR>/bin/<archU>/getapp <appSR>
   ```
   where <appSR> is the full path name to the new version of appSR and <archU> is the Unix architecture.
   ```
   %doGets
   %apCreateTop
   %apCreateApp
   %apCreateIocName
   %apFixLinks
   %gmake doFixRptDct
   ```
   Note: DCT `rpt` files must NOT be out for `sccs edit`. `gmake doFixRpt` may be run instead of `gmake doFixRptDct` if `gmake doFixRptDct` has been run once in the application system area.
5. The application system area is rebuilt by the command:
   ```
   %gmake world
   ```
   See section "Building A Single Application" on page 14 for what this command does.
6. After the application system area has rebuilt successfully, application developers can resync their shadow nodes as described in section "Synchronizing An Application Shadow Node" on page 9.

4. Shadow Node Procedures

This section describes procedures that apply only to an application shadow node. An application shadow node is an image of a complete application system area. When created it contains soft links to files in the application system area. The application developer should perform all development in a shadow area rather than the system area so that other developers do not see his/her changes until `sccs deltas` are executed.

Creating An Application Shadow Node

```
%mkdir <shadow>
```

NOTE: Do not do this in an application system area.

```
%cd <shadow>
%<top>/appSR/bin/sun4/apCreateShadow  <top>
```

You will be asked to create a file with the touch command.

```
%touch  .applShadow
%<top>/appSR/bin/sun4/apCreateShadow  <top>
```

An application shadow area is identical to an application system area with the following exceptions:

1. All files in an application shadow node are initially links.
2. All directories in an application shadow node are real except for each SCCS directory which is a symbolic link.
3. DCT `Db` directories have only the `<*>Db.database` file as a link initially.

Synchronizing An Application Shadow Node

Any time the application system area is rebuilt or changed the application developer must synchronize his or her application shadow node.

```
%cd <shadow>
```

Repeat the following three lines until the status is correct.

```
%apStatusSync
%<edit apRemoveScript via your favorite editor>
%apRemoveScript
```

Repeating the above lines removes the out-of-date files and directories.

```
%apCreateShadow <top>
```

The `apStatusSync` tool is designed to be invoked one or more times before producing the correct `apRemoveScript` file. Status reports go to standard out and commands to remove shadow area components are placed into the `apRemoveScript` file. `apStatusSync` should be repeated until it runs successfully. It is the application developer's responsibility to determine when the status is correct. `apRemoveScript` contains a set of (commented out) Unix commands to remove obsolete or illegal files and/or directory components in the application shadow area. It is the application developer's responsibility to edit the `apRemoveScript` file.

5. Adding/Modifying Components

This section describes procedures for adding or modifying application components. These procedures will work in the either the application system area or in a shadow area. If they are issued in the system area remember that all users may be affected. Wherever this chapter refers to `<shadow>` it is also possible to use `<top>`.

IOC Databases

The procedures given in this section assume that the user is in a database directory, i.e. one of the following commands has been issued:

```
%cd <shadow>/<app>App/<db>Db
```

OR

```
%cd <top>/<app>App/<db>Db
```

NOTE: It is recommended that GDCT, rather than DCT, be used to create and modify database files.

GDCT Databases

Refer to the GDCT User's Manual for details. For each database the following files exist:

- **<dbname>** The file containing graphical information.
- **<dbname>.db** A loadable ASCII file used by GDCT.

All new `<dbname>` and `<dbname>.db` files must be placed under source release control via `sccs create` commands. The `sccs edit`, `sccs get`, and `sccs delta` commands may be issued as necessary.

It is recommended that GDCT databases are loaded with the `dbLoadRecords` and `dbLoadTemplate` commands rather than `dbLoad`.

DCT Databases

DCT can be used to create new databases and/or modify existing databases. This subsection describes application source/release tools that allow the `.rpt` files to be placed under source/release control.

DCT causes a problem for source/release control because DCT generates many files for a single database. What is put under `sccs` control is the short form report file, which must have a file extension of `.rpt`.

For each database in a database directory the following files can exist:

- **<dbname>.rpt** File that is placed under source/release control.
- **<dbname>.rpt0** File generated by the `gmake` utility described below. If such files appear after running `gmake`, the user should resolve the differences and make sure that only `<dbname>.rpt` remains.
- **<dbname>.rpt1, etc** If the user runs `gmake` without resolving differences then `make` keeps creating new files.
- **<dbname>.rpt.err** After checking for real errors these can be deleted.
- **<dbname>Db.database** File generated by DCT. The `gmake` utility described below automatically adds "`Db`" to the report file name.
- **<dbname>Db.ai, etc** The other files generated by DCT.

Report files and Source/Release Control

All report files should be managed via the `dbsccs` commands described in section "dbsccs For DCT Databases" on page 11.

Update Databases From .rpt Files That Are Under sccs Control

```
%gmake
```

For each `<dbname>Db.database` file with a report file that is under `sccs` control, `gmake` performs the following steps when the `<dbname>Db.database` is out of date with respect to the `<dbname>.rpt`, `default.dctsdr`, or `default.sdrsum` files.

1. If `<dbname>Db.database` does not agree with its associated `<dbname>.rpt` then a new short form report `<dbname>.rpt0` is generated (NOTE: If this file already exists it uses `<dbname>.rpt1`, etc). In this case a warning message is also issued.
2. It deletes existing `<dbname>Db.*` files and uses `atdb` to create a read only `<dbname>Db.database` file from the `<dbname>.rpt`.

Note that the new `.database` file agrees with the original `.rpt` NOT the `.rpt0` file.

dbsccs For DCT Databases

Each database is normally represented as a single link to the `<dbname>Db.database`. This eliminates a lot of clutter in the DCT database directory. The `dbsccs` tool allows an application developer to edit specific databases.

DCT should be used to create a new `<dbname>Db` database, and `dbsccs create` should be used to create the `<dbname>.rpt` file and put it under `sccs` control. Once the `<dbname>.rpt` is under `sccs` control, DCT can be used to modify the `<dbname>Db` database only when the `<dbname>.rpt` file is out for edit.

create

```
%dbsccs create <dbname>.rpt
```

This command performs the following functions:

- If `<dbname>.rpt` is already under `sccs` control the command aborts.
- If `<dbname>Db.database` is missing, a link or not writable the command aborts.
- If a `<dbname>.rpt` file already exists, it is renamed `<dbname>.rpt0`. (NOTE: If this file already exists it uses `<dbname>.rpt1`, etc)
- `dbta` is invoked to create a `<dbname>.rpt` file.
- `sccs create` is invoked on the new `<dbname>.rpt` file. The writable `<dbname>Db` database files are removed.
- `atdb` is used to create a read only `<dbname>Db`.database file from `<dbname>.rpt`.

edit

```
%dbsccs edit <dbname>.rpt
```

This command performs the following functions:

- If `<dbname>Db.database` is writable the command aborts.
- `sccs edit` is used to take `<dbname>.rpt` out for edit.
- The `<dbname>Db.*` files are removed.
- DCT is invoked to create a writable `<dbname>Db` database.

NOTE: If a writable `<dbname>Db` database exists, DCT can be used to delete the database before issuing `dbsccs edit`.

delget

```
%dbsccs delget <dbname>.rpt
```

This command performs the following functions:

- If `<dbname>.rpt` is missing, a link or not writable `dbsccs` aborts
- `dbta` is invoked to create `<dbname>.rpt` file from the `<dbname>Db` database.
- The writable DCT `<dbname>Db` database files are removed.
- `atdb` is invoked to create a read only `<dbname>Db.database` file.
- `sccs delget` is invoked for `<dbname>.rpt`.

unedit

```
%dbsccs unedit <dbname>.rpt
```

This command performs the following functions:

- `sccs unedit` is invoked on `<dbname>.rpt`.
- A backup `rpt` is made if `<dbname>Db.database` was modified.
- The writable `<dbname>Db`.database files are removed.
- `atdb` is used to create a read only `<dbname>Db`.database file from `<dbname>.rpt`.

IOC Configuration Files

For each IOC an `ioc` directory exists under `<top>/ioc`. In each such directory a vxWorks startup file exists for each supported vxWorks board support package. In addition resource files can also be placed under source/release control.

vxWorks Startup Files

The startup files (for example `<top>/ioc/<iocname>/st.cmd<archV>`) must be modified after initial creation and when the set of databases to be loaded changes. The normal `sccs edit` and `sccs delta` commands should be used as necessary. When initially created the startup files are prototypes, which contain modification instructions.

resource.def Files

The `iocInit` command in the startup file can have an optional “`resource.def`” parameter. If it does, then the `resource.def` file is processed. This file should appear in the same directory as the startup file. It should be placed under source/release control with the `sccs create` command. Commands `sccs edit` and `sccs delta` can be used as necessary.

ASCII Definition Files

All applications under `<top>` must share the same set of ASCII definition files. Two directories are available for application ASCII files. `<top>/replace_ascii/` is the place to store files that are replacements for EPICS files and `<top>/cat_ascii/` is the place to store files that don’t exist in EPICS and files that should be added to the end of EPICS files.

The command `makesdr` must be run any time there is a change to any ASCII input file used by `makesdr`. After `makesdr` completes all databases must be rebuilt and any affected record or device support must be rebuilt.

After `makesdr` is executed all applications must be rebuilt. The following commands will rebuild all applications:

```
%cd <top>
%gmake
```

This will rebuild all out-of-date applications.

If, however, you are working in a shadow area and are only dependent on a single application, the following commands can be used to rebuild the single application:

```
%cd <shadow>
%makesdr
%cd <app>App
%gmake
```

Adding Application Specific Source Files

All application specific source files are put in `<top>/<app>App/src`. When any new file is placed in this directory it should be put under source/release control via the `sccs create` command. Depending on the type of file other files will have to be edited.

The Makefiles for building application specific source files have the same structure and features as the Makefiles described in the *EPICS Source/Release Control: How to Build and Develop EPICS Software* document. See Chapter 6 in that document for information on creating and using Makefiles.

NOTE: The Imakefile method is provided for compatibility. Users are encouraged to use the Makefile method.

Unix Source Files (Makefile method)

This includes C sources and include files.

```
%cd <src>
%<EDIT>Makefile.Unix
```

Edit this file to build the new Unix component.

```
%gmake
```

Components are built in the `O.<archU>` subdirectories.

```
%gmake depends
```

A `.DEPENDS` file containing dependency information will be built in the `O.<arch>` subdirectories.

The following makefile targets are supported:

- `clean`, `build` (default) and `depends`, for all site supported architectures, and
- `clean.<arch>`, `build.<arch>` and `depend.<arch>` for a single architecture.

IOC Source Files (Makefile method)

This includes C sources, include files and sequence programs. If the sources are for record, device, or driver support remember that ASCII definition files must be prepared and `makesdr` executed as described above. In addition the following must be performed:

```
%cd <src>
%<EDIT>Makefile.Vx
```

Edit this file to build the new IOC component.

```
%gmake
```

Components are built in the `O.<archV>` subdirectories.

```
%gmake depends
```

A `.DEPENDS` file containing dependency information will be built in the `O.<arch>` subdirectories.

The following makefile targets are supported:

- `clean`, `build` (default) and `depends`, for all site supported architectures, and
- `clean.<arch>`, `build.<arch>` and `depend.<arch>` for a single architecture.

Unix Source Files (Imakefile method)

This includes C sources and include files.

```
%cd <src>
%<EDIT>ImakefileUnix
```

Edit this file to build the new Unix component.

```
%cd <app>App/<archU>
%Buildit
%gmake
```

IOC Source files (Imakefile method)

This includes C sources, header files and sequence programs. If the sources are for record, device, or driver support remember that ASCII definition files must be prepared and `makesdr` executed as described above. In addition the following must be performed:

```
%cd <src>
%<EDIT>ImakefileVx
```

Edit this file to build the new IOC component.

```
%cd <app>App/<archV>
%Buildit
%gmake
%gmake depend
```

Modifying Existing Application Sources

In this case just edit the source in `<src>` and then execute either the `gmake` command in the `<src>` directory (Makefile method) or the `gmake` command in either the `<archU>` or `<archV>` directory (Imakefile method).

Operator Files

All files placed in the `op` directory should be managed via the `sccs` commands.

Building A Single Application

```
%cd <shadow>/<appName>/
%gmake
```

This does a `gmake` in each defined `<archV>` and `<archU>` directory, in the `src` directory, and in each `<*>Db` directory. If a "`makefile.pvt`" makefile exists it is then invoked.

Individual application components can be rebuilt by qualifying the `gmake` command:

```
%gmake doGets
```

This brings all `sccs` files in this directory and below up-to-date.

```
%gmake bldDb
```

This recreates each database as described in section "IOC Databases" on page 10.

```
%gmake build
```

This performs a `gmake build` in the `src` directory (Makefile method).

```
%gmake depends
```

This performs a `gmake depends` in the `src` directory (Makefile method).

```
%gmake bldMakefiles
```

This performs a `Buildit` and `gmake` in each defined `<archV>` and `<archU>` directory (Imakefile method).

```
%gmake bldPvt
```

If a file named `makefile.pvt` exists then a `gmake` is performed using this file.

All of the above can be performed by issuing the command:

```
%gmake world
```

6. Source/Release Tools

Application System Area

This section describes tools that should be issued only in an application system area, NOT in a shadow area.

Tools invoked in the <top> directory

getrel: This command is executed in `<top>` to get a new release of EPICS. It is always issued in `<top>`. When issued it must be executed with a full path name to the release of EPICS desired.

getapp: This command is executed in `<top>` to get a new release of appSR. It is always issued in `<top>`. When issued it must be executed with a full path name to the release of appSR desired.

apCreateTop: This command must always be issued in directory `<top>`. The first time this command is issued, it creates all directories and files needed for an application system area. It must also issued whenever a new release of epics is obtained via the getrel command. In this case it makes sure that application system area is correct for the new release.

apCreateApp: This command creates the directories needed for each application that resides under `<top>`. It is executed in `<top>` whenever new applications are added to file `<top>/appList`.

apCreateIocName: This command creates the directories needed for each IOC that resides under `<top>/ioc`. It is executed in `<top>` whenever new IOCs are added to file `<top>/iocList`.

apFixLinks: This command will regenerate generic links.

apFixDctRpt: This command will convert DCT short form reports to `dbta` report format. Executing `apFixDctRpt` with parameter `dct` will force use of `dct` instead of `atdb` to read the short form report.

Tools invoked in the <top>/<app>App/ directory

apCreateDbDir: This command creates the database directories used by a particular application. It is executed in `<top>/<app>App` and is an interactive tool.

Tools invoked anywhere

doGets: This command uses make to ensure that all `sccs` controlled files in this directory and below are up-to-date.

Application Shadow Area

This section describes tools that only apply to a shadow area.

apCreateShadow: The first time this command is executed in a directory it creates a complete shadow area. It is also issued to fill in missing links whenever the application system area has been rebuilt.

apStatusSync: This command is issued whenever the shadow area must be resynced with the system area because the system area was rebuilt. It issues error messages to standard out and also writes Unix commands into a file `apRemoveScript`. If it reports errors the user should fix the errors and reissue the `apStatusSync` command. When the user is satisfied then the apRemoveScript must be edited and executed. apRemoveScript contains a number of rm commands but they are commented out (preceded by #). The user should decide which files should really be removed.

dbsccs: A tool for managing databases in a shadow directory. See section "dbsccs For DCT Databases" on page 11 for details

Development Tools

This section describes tools that can be issued in either a system area or in a shadow area. They are issued in shadow areas during development and in the system area during integration.

apSccsInfo: This tool will search all directories below the current directory and list all sccs controlled files that are currently out-for-edit.

makesdr: The purpose of the makesdr tool is to allow application developers to build a private default.dctsdr file. makesdr allows application developers to modify, by appending to or replacing, any EPICS ASCII definition file used in creating the default.dctsdr file. The default.dctsdr file is required by DCT/GDCT. The makesdr tool also allows new ASCII definition files to be introduced into the application environment.

makesdr first searches the EPICS ascii directory followed by the cat_ascii directory and then the replace_ascii directory in order to determine the composition of each ASCII definition file. The composed ASCII files are then processed by cpp and the various SDR "bld" tools in order to produce SDR structures for the default.dctsdr file, record header files, and the default.sdrSum file. The end result of a successful makesdr run is that EPICS record header files etc. are either replicated or updated into a new sdrH/rec directory. makesdr corrects any EPICS files according to the contents of the local ASCII directories. If the default.dctsdr or default.sdrSum files/links changed they are replaced with the new versions. ASCII files placed in the replace_ascii directory supercede all other ASCII input files with the same name. The application developer is expected to include the sdrH/rec directory when doing a vxWorks build.

Directory sdrH/rec/ is created from scratch the first time makesdr is run. It contains either copies of EPICS header files or the versions created by makesdr.

Note: makesdr only rebuilds if something is out-of-date.

Buildit: This command creates a Makefile from an Imakefile. Whenever an Imakefile is modified Buildit must be executed. Note that the Imakefiles are stored in <src>, but Buildit is executed in <archV> for vxWorks and in <archU> for unix.

gmake

This command rebuilds various application components. What it does depends on where it is executed.

- **<top>**
  - **gmake doFix**: Runs apFixLinks
  - **gmake doFixRptDct**: Runs apFixDctRpt dct
  - **gmake doFixRpt**: Runs apFixDctRpt
  - **gmake doGets**: Runs doGets
  - **gmake domakesdr**: Runs makesdr
  - **gmake bldMakefiles**: Rebuilds Makefiles from the Imakefiles, runs "gmake clean" and then "gmake" for each Imake architecture in each application
  - **gmake doapplications**: Runs "gmake" on the makefile in each application

- **gmake doappworld**: Runs "gmake world" on the makefile in each application
- **gmake world**: Does all of the above
- **gmake**: Defaults to "gmake doapplications" above.
- **gmake dotar**: Creates a compressed tar file in the directory above <top> and names the file **"**<top>.Tar.Z**".** Ex: if this was the <top> directory for par the file would be named par.Tar.Z.
- **gmake tarinfo**: Displays directions for unpacking the compressed tar file.
- **gmake clean**: Runs "gmake clean" in the src directory of each application to remove all temporary files from the O.<arch> directories.
- **gmake build**: Runs "gmake build" in the src directory of each application to compile and link objects. The objects reside in the O.<arch> subdirectories.
- **gmake depends**: Runs "gmake depends" in the src directory of each application to create, in each O.<arch> subdirectory, a .DEPENDS file of header file dependencies.

- **<shadow>**: The same as except that apFixLinks is a NOP.
- **<app>App**: See section "Building A Single Application" on page 14 for what make in an application directory does.
- **<db>Db**: The actions described in section "Update Databases From .rpt Files That Are Under sccs Control" on page 11 are performed.
- **<archU>**: The Unix components are rebuilt (Imakefile method).
- **<archV>**: The IOC components are rebuilt (Imakefile method.
- <**src**>: The Unix and IOC components are rebuilt (Makefile method).
- **<src>/O.<arch>**: The <arch> components are rebuilt (Makefile method).

7. APPENDIX Application System Area

An application system node contains the following files and directories.

```
<top>
 ├── .current_rel_hist      ┐
 ├── ascii/@                │
 ├── base/@                 │
 ├── config/@               │
 ├── epicsH/@               ├ Links to EPICS components
 ├── include/@              │
 ├── targetmv167/@          │
 ├── vw/@                   │
 ├── vxWorksmv167@          │
 ├── vxWorksmv167.sym@      ┘
 │
 ├── default.dctsdr@
 ├── default.sdrSum@
 │
 ├── cat_ascii/             ┐
 ├── replace_ascii/         ┘ ASCII files
 │
 ├── appSR/@                ┐
 ├── makefile@              ┘ Links to appSR components
 │
 ├── iocList
 ├── ioc/
 │    ├── (common_ioc_files)
 │    └── <iocName>/          A directory for each IOC name in the iocList file
 │         ├── st.cmd<archV>
 │         ├── resource.def
 │         └── vxWorks<archV>@
 │
 ├── appList
 └── <appName>App/            A directory for each application in the appList file
```

Each <app>App directory contains the following files and directories.

```
<appName>App/
 |
 |-- makefile@
 |
 |-- src/
 |    |-- (source files)
 |    |
 |    |-- ImakefileVx    \
 |    |-- ImakefileUnix  /  Used for Imakefile method
 |    |
 |    |-- Makefile       \
 |    |-- Makefile.Unix   > Used for Makefile method
 |    |-- Makefile.Vx    /
 |    |
 |    |-- O.<archV>/          A directory for each Vx arch (Makefile method)
 |    |    `-- (object files)
 |    `-- O.<archU>/          A directory for each Unix arch (Makefile method)
 |         |-- (object files)
 |         `-- (executables)
 |
 |-- archList
 |-- <archV>/          A directory for each Vx arch in archList (Imakefile method)
 |    |-- Imakefile@
 |    `-- Makefile
 |-- <archU>/          A directory for each Unix arch in archList (Imakefile method)
 |    |-- Imakefile@
 |    `-- Makefile
 |-- <db>Db/                  A directory for each dbName
 |    |-- makefile@              (gdct or dct type makefile)
 |    |-- default.dctsdr@
 |    |-- default.sdrSum@
 |    |-- ( <dbName>.rpt files )            ( present in dct type directory only)
 |    |-- ( <dbName>.databsase files )      ( present in dct type directory only)
 |    |-- ( <dbname>.db files)              ( present in gdct type directory only)
 |    `-- ( <dbName> files)                 ( present in gdct type directory only)
 |
 `-- op/
      |-- adl/
      |    `-- (adl files)
      |-- alh/
      |    `-- (alh config files)
      |-- ar/
      |    |-- arChan/
      |    |    `-- (ar chan files)
      |    |-- arReq/
      |    |    `-- (ar req files)
      |    `-- arSet/
      |         `-- (ar set files)
      |-- burt/
      |    `-- (burt request files)
      `-- km/
           `-- (km config files)
```

8. APPENDIX Application Production Area Evolution

One problem that still has to be addressed is generation of a production area for use by operations. This appendix presents a possible set of procedures that could be used. Please refer to the next page for a flow diagram architecture.

1. The application manager creates and maintains the application system area (A:). This would include changing to a new EPICS release.
2. Application developers create and maintain shadow areas accessing the application system area (A:). Deltas are applied to the application system area to make changes permanent.
3. The production manager requests a new version of the application system area.
4. The application system manager fulfills this request by replicating the application system area (A:) into the application integration area (B:) and running a baseline set of regression tests.
5. The Unix system manager changes ownership of the application integration area (B:) to the application production manager.
6. The application production manager retires the (D:) previous production area.
7. The application production manager moves the current application production area (C:) to the previous production area (D:).
8. The application production manager moves the application integration area (B:) to the application production area (C:)
9. The application production manager deletes and recreates a production shadow area (E:) to be used for quick fixes.