Inter-Office Memorandum

To	Cedar Implementors	Date	May 26, 1982
From	Roy Levin	Location	Palo Alto
Subject	Cedar Releases: Policies and Procedures	Organization	PARC/CSL

XEROX

Filed on: [Indigo]<Cedar>Documentation>ReleaseProcedures.press

This memo describes the policies and procedures that individuals contributing to Cedar releases need to understand and observe. It assumes general familiarity with Cedar and DF files.

The Release Process

Components and DF Files

Throughout this memo, we call the unit of software submitted to the Cedar release process a *component*. Each component is described by a *DF file*, a simple description mechanism that is chiefly a list of file names (complete IFS paths) and creation dates (either explicit or implicit). This memo assumes familiarity with DF files; complete information may be found in the reference manual stored on [Indigo]<Cedar>Documentation>DFFilesRefMan.press. On occasion, a particularly large or complicated component may be more easily described by a collection of DF files. This is acceptable, but there must be a single "root" of the collection which includes (in the technical sense) the rest. A DF file contains information that describes the component for three distinct but related purposes. First, it identifies everything needed by the implementor to build the component from scratch. Second, it identifies the subset of files that are necessary for a client of the component. Third, it describes the way in which the files that make up the component are to be distributed at release time. It is the responsibility of the implementor to ensure that a DF file intended for release as part of Cedar fulfills all three of these descriptive functions; this memo contains rules that help him in doing so.

Release Phases

A release cycle consists of a *development phase* following by an *integration phase* culminated by a *release*. There is an individual, the Release Master, who monitors the release cycle and is responsible for the orderly progression from one phase to the next. The development phase lasts for days or weeks and is a period of largely unconstrained program modification and enhancement. The scope and duration of the development is release-dependent and is generally not completely specified when the development phase begins, although each implementor generally knows approximately what he intends to contribute to the next release. At some point, the Release Master initiates the integration phase and the following sequence of events then occurs:

1) The Release Master sends a message to CedarImplementors^.pa announcing the beginning of the integration phase. Obviously, you must be a member of CedarImplementors^.pa to receive this message; if you are a new implementor, you should verify that you are on this list. The message requests implementors to complete the development of their release components and submit them by responding to the Release Master with a specific

message form. Both the contents of the form and the submission itself must satisfy a number of rules before the Release Master will accept the submission for inclusion in the upcoming release. These rules are presented in detail later in this memo. The message announcing a submission is also sent to CedarImplementors^.pa so that other implementors can complete the preparation of their components.

A component cannot be submitted to the Release Master until all of the components it depends upon have been announced. Experience shows that, if the release is expected to contain changes to fundamental interfaces (e.g., Pilot, Rope, IOStream), the Release Master should probably send the "call for integration" message only to the implementors of those central components. Once these submissions have been accepted, the Release Master can then send the "call for integration" to all implementors.

2) The Release Master assembles a top-level description of the contents of the release and runs the Release Tool. This program has three phases, the first two of which check the release submissions for consistency and completeness. The third phase moves the files to the release directory and produces descriptions (DF files) of the release contents. Generally, however, the Release Tool will detect some problems during the execution of its first two phases.

3) The Release Master analyzes the error messages produced by the initial phases of the Release Tool and contacts the implementors of the components that are in error. *The integration phase of the release cycle is stalled until the implementors correct their erros;* consequently, it is the responsibility of the implementors to do so as soon as possible. They notify the Release Master when they completed the necessary repairs.

4) Steps 2 and 3 are repeated until the Release Master is satisfied of the consistency and completeness of the submissions. The Release Master then sends a message to CedarUsers^.pa announcing that a release is imminent and that the release directory will be in an inconsistent state while the release is installed.

5) The Release Master runs the third phase of the Release Tool, which copies the release files to the release directory. The Release Master tries to make this phase as brief as possible, anticipating potentially disabling problems (e.g., insufficient disk space on the release directory) and taking the necessary precautions.

6) When the release has been completely installed, the Release Master sends a message to CedarUsers^.pa announcing the release. The Release Master should compose this message during step 5 in order to minimize the time during which the release directory is inconsistent. The Release Master stores a copy of the release message on the documentation subdirectory of the release directory, then prints and saves the logs produced by the Release Tool. The release is then complete.

The Need for Rules

A Cedar release typically includes thousands of files and hundreds of DF files, the latter with a complex interconnection structure. It is not surprising that errors are committed during the integration phase that force steps 3 and 4 to be repeated several times. Attention to the details of preparing a release submission can significantly reduce the duration of the integration phase. Experience shows that the three phases of the Release Tool can be completed in a few hours for a major release, but, nevertheless, such releases generally take two days. The rest of the time is chiefly spent waiting for implementors to correct errors in their submissions. It is important to understand that normally these are not programming bugs in the usual sense; rather, they are errors in the DF files that describe the components. Certain kinds of errors occur again and again, release after release, and the rules below are intended to eliminate these recurring problems.

These rules should be considered stronger than guidelines or suggestions but somewhat weaker than commandments. Uniformity promotes smooth and speedy releases, but exceptional cases invariably arise that do not fit the general model. Occasional reasonable deviations will be tolerated, but the Release Master establishes the definition of "reasonable". Unreasonable deviations jeopardize the release process and cannot be permitted. In all cases the implementor assumes complete responsibility for any deviation from these rules. The Release Master will not correct errors in submissions and will delay a release as necessary until an offending submission conforms to the requirements below.

Rules for Constructing a Cedar Release DF File

The following is a complete list of requirements that a DF file must satisfy in order to be acceptable as a component of a Cedar release. Unless explicitly indicated otherwise, the terms "import" and "export" refer to the Imports and Exports clauses in a DF file and not the similarly-named Cedar language notions. Also, the phrase "released to [X]<Y>Z>" is an abbreviation for "has a ReleaseAs clause with path [X]<Y>Z>".

1) *The DF file name is considered to be the "name" of the component and should generally correspond to the name of the major exported interface (in the Mesa sense) supplied by the component.* This rule is most easily applied to straightforward packages with a single public interface.

2) *The DF file must contain a self--reference that is released to [Indigo]<Cedar>Top>. The self-reference must be exported.*

3) *If separate documentation files accompany the component, they should be included in the DF file and released to [Indigo]<Cedar>Documentation>. Documentation files in suitable form for online use (not press files) should also be exported.* It is not mandatory that documentation files accompany a component; comments in the public interfaces may be sufficient for use. However, most substantial components will have separate documentation files, and these should be included in the DF file as indicated. If a component has recently undergone a substantial revision that is visible to its users, the implementor should probably release separate files containing the "change summary" and the "complete truth".

4) *All component-specific files other than the DF file and documentation should be released to a subdirectory with the same name as the component (e.g., the interfaces and implementation of SomePackage.df should be released to [Indigo]<Cedar>SomePackage>*. It is permissible to have additional subdirectory structure within the component-specific subdirectory. Subdirectories of either [Indigo]<Cedar>Top> or [Indigo]<Cedar>Documentation> are not permitted.

5) *If the component supplies a "public interface", the source and object files must be exported from the DF file.* The debugging and help facilities tend to work better if they can access the sources of interfaces as well as their compiled representations. Disk space concerns are irrelevant here, since the files involved are small and improvements in the file system will soon eliminate the size limitations of the local disk.

6) *The BCD or BCDs that contain the bound-up implementation of the component must be exported from the DF file. Exception: if the implementation is intended for inclusion in the Cedar boot file, it should not be exported.* The motivation here is obvious; the component can't be used unless its implementation is available.

7) *The entry in the DF file for the bound-up implementation of the component must be preceded by a "+" to indicate that it is a root BCD for processing by VerifyDF.* If the component exports multiple BCDs, each of them must obey this rule. It is permissible to have other BCDs marked with a "+", e.g., test programs.

8) *A file that is not a part of the component should be referenced using an "Imports" entry with a USING list.* The entry forms introduced by "ReadOnly" and "@", while documented in the DF reference manual, are obsolete and should not be used. The USING list documents explicitly dependencies on other components, protects against certain mistakes in imported DF files, and speeds up BringOver.

9) *An import from the release directory may include either an explicit or an implicit (i.e., ">" or "~=") date. An import from a working directory must specify an implicit date.* While these rules may seem a bit unintuitive at first, experience shows that they greatly streamline the release process. Last-minute, minor changes in a DF file are common during a release integration, and a failure to observe the second rule in a DF file depending on the one that changed forces an otherwise unnecessary reconstruction of the dependent DF file. The last-minute changes are rarely significant to the importer(s); when they are, inconsistencies will detected by the release machinery.

10) *The DF file for a component should usually include a text file containing command lines for compiling and binding the package.* This is not mandatory, but strongly recommended. It is not necessary that this file be a complete command file; rather, it is intended more as an documentary aid to someone unfamiliar with the structure of the component who finds it necessary to make a small change and rebuild it. There is no standardization on the name of such text files; most have the extension ".cm" (whether or not they are true command files) and many include the word "Make" and the component name in their names.

11) *If programs other than the Compiler and Binder are needed to build the component, entries importing them should be included in the DF file.* Examples of such programs include the Packager, MakeBoot, and the TableCompiler. If specific versions of the Compiler and/or Binder are required (a rare situation), entries for them should be included as well; however, it is not necessary to reference the standard ones.

12) *The working directory used for all non-imported files must grant read and create access to CedarAdministrators^.PA.* The need for read access is obvious; the component can't be released unless it can be read. The need for create access is rarely exercised but is occasionally needed when a small change to a component is required in order to complete a release integration and the implementor is not available. Note that the heavily-used working directories [Indigo]<CedarLang> and [Indigo]<CedarLib> satisfy these requirements, while personal directories typically do not. Implementors who use their personal directories for release submissions must explicitly alter the IFS access controls to their directories to conform to this rule.

A Sample DF File

The DF file below satisfies the requirements for submission to a Cedar release. It describes a component named `Sample`, which consists of a single configuration named `SampleImpl`. `Sample` has two public interfaces, named `Sample` and `SampleExtras`. The implementation consists of two modules, `SampleImplA` and `SampleImplB`, which share a private definitions module named `SamplePrivate`. It requires three other components, defined by DF files named `PilotInterfaces.df`, `Rigging.df`, and `Runtime.df`. Finally, the component is documented by a press file whose source is a Tioga document.

Notations of the form {n} are not actually part of the DF file. They indicate that the entry so marked conforms to rule n, above.

```
// Sample.df
// last edited by Harry Bovik: May 10, 1982  5:26 PM

Exports [Ivy]<Bovik>Sample>     ReleaseAs [Indigo]<Cedar>Top>   {2}

  Sample.df!3                                10-May-82 11:47:21 PDT  {1}

Exports [Ivy]<Bovik>Sample>     ReleaseAs [Indigo]<Cedar>Sample> {2}

  Sample.bcd!2                                9-May-82 17:45:27 PDT {3}
  Sample.mesa!2                               9-May-82 17:44:55 PDT {3}
  SampleExtras.bcd!1                         10-May-82 11:45:31 PDT {3}
  SampleExtras.mesa!1                        10-May-82 11:33:19 PDT {3}

  +SampleImpl.bcd!7                          10-May-82 14:53:09 PDT {6,7}

Directory [Ivy]<Bovik>Sample>     ReleaseAs [Indigo]<Cedar>Sample> {4}

  SampleImpl.config!2                        10-May-82 13:24:52 PDT
  MakeSample.cm!3                           10-May-82 13:25:19 PDT {11}

  SamplePrivate.bcd!2                        10-May-82 12:36:12 PDT
  SamplePrivate.mesa!1                       10-May-82 12:34:51 PDT

  SampleImplA.bcd!7                          10-May-82 14:47:39 PDT
  SampleImplA.mesa!6                         10-May-82 14:47:21 PDT
  SampleImplB.bcd!4                          10-May-82 14:47:43 PDT
  SampleImplB.mesa!2                         10-May-82 14:32:06 PDT

Exports [Ivy]<Bovik>Sample>  ReleaseAs [Indigo]<Cedar>Documentation> {4}

  Sample.press!2                              6-May-82  9:40:18 PDT
  Sample.tioga!2                              6-May-82  9:37:23 PDT

Imports [Indigo]<Cedar>Top>PilotInterfaces.df!1 Of 1-Apr-82 15:43:56 PST
    Using [Environment.bcd, Process.bcd] {8,9}

Imports [Indigo]<Cedar>Top>Rigging.df!9 Of 7-May-82 20:18:46 PDT
    Using [Rope.bcd] {8,9}

Imports [Indigo]<CedarLang>Runtime>Runtime.df Of >
    Using [SafeStorage.bcd] {8,9}
```

Notice that the files that make up the component are all stored on a working directory chosen by the implementor, in this case his personal directory on Ivy. (Note that, in accordance with rule 12, Bovik must grant create access on [Ivy]<Bovik> to CedarAdministrators^.pa.) The DF file specifies the locations on a release directory, [Indigo]<Cedar>, where the files are to be stored at release time. The component depends on three other components, two of which have been previously released (since they are on the release directory) and one of which (`Runtime.df`) is not (and presumably will be released when `Sample` is).

Rules for Submitting a Component for Release

The message from the Release Master that announces the integration phase of a release contains a message form. This form, properly completed and returned, serves three important functions:

1) It enables the Release Master to include the DF file describing the component in the top-level description of the release.

2) It notifies other implementors that the component is now available for reference in other DF files.

3) It provides the information to be included in the release message that will announce the release to all Cedar users.

The message form has been constructed to facilitate all three of these purposes. Since deviations from the standard form complicate and delay the integration phase, implementors must observe the following rules for submission of components:

13) *The component must satisfy the construction rules before the submission form is sent.* Sending the form before the DF files are ready and stored confuses everyone.

14) *The implementor must have run VerifyDF on the component's DF file before submitting it for release.* VerifyDF performs most of the completeness checks that the Release Tool's second phase does; consequently, most errors of this form can be detected early in the integration phase and without delaying other implementors.

15) *The submission form is the only acceptable way of announcing a release submission.* Free-form messages are not acceptable and will be rejected by the Release Master. If an implementor expects to be unavailable during the integration phase and wishes to prepare his submission "in advance", it is his responsibility to obtain from the Release Master a copy of the form and use it.

16) *All fields of the form must be completed appropriately.* In particular, the working path and release path must be filled in correctly and expressed in conventional syntax (server name in square brackets, directory and subdirectories in angle brackets); "/" syntax is not acceptable. The documentation pointer must conform to the requirements stated on the form. The portion of the form that contains information destined for the release message should be filled in judiciously; it is generally not necessary or desirable to describe every bug fix and minor enhancement. The reference documentation should include this kind of information, while the release message should contain a summary of the significant changes in the component since the last release. A chronology of changes is *not* appropriate. The release message portion of the form must also be coherent English prose that observes accepted standards of capitalization, spelling, punctuation, and grammar.

Avoiding Common Mistakes

When the integration phase of a release cycle is underway, it is easy to forget one or more of the above rules. Experience has shown that some rules are more likely to be broken than others and that certain mistakes occur repeatedly. Careful observation of the following rules will help implementors avoid the most common errors that delay release integrations.

17) *When beginning development of a component following a release, use the released DF file as the base, not the working DF file supplied to the previous release.* Obsolete working DF files are perhaps the single greatest source of trouble in the integration phase. When an implementor uses a working DF file from a previous release as the base for new development, he invites inconsistencies that cannot be easily detected until the Release Tool is run. The typical problem is a reference to a working DF file that is either obsolete or non-existant. SModel can only detect the latter of these problems, and then typically only if /v is specified.

18) *Be certain that all imports come from the correct location (working directory or release directory).* In a sense, this rule includes the preceding one, since incorrect imports frequently arise when an obsolete DF file has been used as the starting point for development of the component.

19) *When correcting a problem uncovered by the Release Tool during the integration phase, always run VerifyDF before notifying the Release Master that the component is again ready for inclusion.* It is quite common for phase one of the Release Tool to uncover a problem in a DF file which, when "fixed" quickly by the implementor, leads to a problem in phase two. This causes unnecessary delay while the implementor once again fixes the problem; running VerifyDF before the first resubmission will nearly always avoid such delays.

20) *Before submitting a component for release, double-check the subdirectories to which files are being released.* The Release Tool cannot, in general, check that rules 2-4 are obeyed. Uniformity here greatly simplifies browsing.

21) *After successfully running VerifyDF (rule 19), be sure to run SModel /v,.* In the course of making a DF file acceptable to VerifyDF, it is useful to run SModel /n, which, although it improves the state of the files on the local disk, performs no transfers to file servers. SModel /v ensures that the cumulative effect of repeated runs of SModel /n is moved to the file servers.