Using the DSPCAD Integrative Command-Line Environment: User's Guide for DICE Version 1.1*

Shuvra S. Bhattacharyya, Chung-Ching Shen, William Plishker,
Nimish Sane, and George Zaki
Department of Electrical and Computer Engineering, and
Institute for Advanced Computer Studies
University of Maryland at College Park, USA
{ssb, ccshen, plishker, nsane, gzaki}@umd.edu

July 4, 2011

This document provides instructions on setting up, starting up, and building DICE and its key companion packages, `dicemin` and `dicelang`. This installation process is based on a general set of conventions, which we refer to as the *DICE organizational conventions*, for software packages. The DICE organizational conventions are specified in this report. These conventions are applied in DICE, `dicemin`, and `dicelang`, and also to other software packages that are developed in the Maryland DSPCAD Research Group [1].

This user's guide is supplemented by an overview of DICE and some of its core features [2], online documentation available in DICE, and various tutorial materials that are available electronically from the DICE User's Guide. Pointers to these resources are available in the Online Supplement [3].

1 What is DICE?

DICE (the *DSPCAD Integrative Command Line Environment*) is a package of utilities that facilitates efficient management of software projects. Key areas of emphasis in DICE are cross-platform operation, support for projects that integrate heterogeneous programming languages, and support for applying and integrating different kinds of design and testing methodologies. The package is being developed at the University of Maryland to facilitate the research and teaching of methods for implementation, testing, evolution, and revision of engineering software. The package is also being developed as a foundation for developing experimental research software for techniques and tools in the area of computer-aided design (CAD) of digital signal processing (DSP) systems. The package is intended for cross-platform operation, and has been developed and used actively on the Linux, Mac OS, Solaris, and Windows (equipped with Cygwin) platforms.

For an overview of key features in DICE, we refer the reader to [2]. This document is intended to supplement this overview with detailed instructions for setting up, starting up and building DICE.

*Technical Report UMIACS-TR-2011-13, Institute for Advanced Computer Studies, University of Maryland at College Park, 2011.

2 Notational Conventions

The following notational conventions are used in this document.

- In-line code fragments within sentences are shown using `this font`. In-line code fragments can be hyphenated across line boundaries, so care should be taken to "filter out" any line-ending hyphens when applying such fragments.
- Line-by-line code fragments are shown

  ```
  using one or more lines that have
  this font.
  ```

- A backslash at the end of a code fragment line indicates that the text on the following line is a continuation of the command on the previous line. For example.

  ```
  gcc -Wall -pedantic -o x.exe \
          a.c b.c c.c d.c e.c f.c
  ```

  For a command that spans multiple lines, we typically indent the code on the continuation lines relative to the code on the first line of the command, as shown above.

- Items in command descriptions that are enclosed by angle brackets (`<...>`) indicate placeholders for command arguments or other text that needs to be customized based on the context of the command.
- Items in command descriptions that are enclosed by square brackets (`[...]`) indicate *optional* command arguments.

3 Setting up your `dice_user` Directory

A `dice_user` directory is a directory in which certain user-specific files associated with one's DICE environment are stored. This directory must be called `dice_user`. A typical location for this directory is `~/dice_user` (i.e., in one's home directory). To set up this directory, just create an empty directory called `dice_user` in the desired location, and then create a subdirectory of `dice_user` called `startup`.

For example:

```
cd ~
mkdir dice_user
cd dice_user
mkdir startup
```

This constructs the following directory structure:

```
~/dice_user/
    startup/
```

It is generally advised that users avoid creating their own *specialized* directories or files in `dice_user` or any subdirectory within `dice_user` except through DICE or `dicelang` commands that create those files or directories as side effects. For example, the `dxbuild` command creates as a side effect some files within `dice_user/dxgen`. Here, by "specialized," we mean files or directories that are not required as part of the standard set up process for DICE or one of its plug-in packages. The one exception to this advice is the `$UXTMP` directory, which is discussed in [2]. The `$UXTMP` directory is intended to be a scratch area for arbitrary user work, including direct (not necessarily through DICE-related commands) creation, manipulation, and deletion of files and directories. Keeping the DICE user tree as "standard" as possible will help to avoid confusion between DICE-related and non-DICE-related user files and directories.

4 Setting up `dicemin`

`dicemin` (the "min" stands for *minimal*) is a very compact subset of DICE that is maintained and installed separately from DICE. `dicemin` should be set up before setting up DICE. The `dicemin` package provides a small, more manageable subset of the overall DICE functionality that is generally easier to troubleshoot (if problems arise) than DICE. Once `dicemin` has been set up properly, the process of setting up the rest of DICE is generally very smooth.

4.1 Downloading `dicemin`

Download `dicemin` (`dicemin.tar.gz`) from the Online Supplement [3]. Then extract the `dicemin/` directory from the `tar.gz` archive in which it is packaged.

4.2 The Standard `dicemin` Startup File

Download the standard `dicemin` startup file (`dicemin_startup`) from the Online Supplement [3]. Place the `dicemin_startup` file in your `dice_user/startup` directory. Typically, this startup file is used "as is," and does not need any editing by the user.

4.3 Essential UX Definitions for `dicemin`

Download the `dicemin` template file `uxdefs_dicemin` for essential "UX definitions". This file is also available from the Online Supplement [3]. "UX" is a prefix used for DICE-related environment variables that correspond to user-specific customizations. Place the file `uxdefs_dicemin` in `dice_user/startup`.

Open the file `uxdefs_dicemin` with a text editor. Set to an appropriate value the right hand side of the assignment statement at the bottom of the file. The meaning of the environment variable setting that is involved in this assignment statement is as follows.

- `UXDICEMIN`: This should be set to the (UNIX/Cygwin-format) directory path of your `dicemin` installation directory.

For example, suppose you are running DICE on Windows, and you have placed your `dicemin` installation in `/home/utils/`. Then your environment variable setting in `uxdefs_dicemin` would be as follows.

```
UXDICEMIN=/home/utils/dicemin
```

Note that on some platforms, you may need to process the `uxdefs_dicemin` and `dicemin_startup` files with the standard `dos2unix` utility before they will execute properly. Such processing just needs to be done once during initial setup of `dicemin`.

4.4 Starting up `dicemin`

To use `dicemin` and any packages, such as DICE and `dicelang`, that depend on `dicemin` in a given `Bash` session, one must first start up `dicemin` within that session. Starting up `dicemin` involves loading necessary environment settings so that you can use all of the features in `dicemin`.

To start up `dicemin`, follow these steps:

1. Start a `Bash` shell.
2. `cd` to your DICE user directory (e.g., run `cd ~/dice_user`).
3. Run

   ```
   source startup/dicemin_startup
   ```

IMPORTANT: The `dicemin_startup` file must be invoked from the `dice_user` directory — e.g., as opposed to running it as

```
source dicemin_startup
```

from `~/dice_user/startup` or running it from your home directory.

If you encounter difficulties starting up `dicemin`, see Section 13.

4.5 Building `dicemin`

To use all of the features available in `dicemin`, you need to build certain executables from the corresponding source code. To do this, simply startup `dicemin`, and then, from the same `Bash` session, run the `dicemin` utility `dxmbuild` (with no arguments). This step of building `dicemin` needs to be done *only once* for a given version/update of `dicemin`.

5 General Setup Procedure

The general process of setting up and using DICE and other packages that are based on organizational conventions employed in DICE is the same as that employed in the process of setting up `dicemin`. This general setup process applies to all packages that are based on DICE organizational conventions, including `dicemin`, DICE, and `dicelang`.

The general setup process is based on the following package-specific parameters.

5.1 Package Parameters

Each parameter is listed along with a pseudocode symbol that helps to identify the parameter succinctly in documentation and examples.

- Package name <PK_NAME>.
- Package user directory <PK_USER_DIR>.
- Package download site <PK_SITE>.
- Package download file <PK_FILE>.
- Package definitions file <PK_DEFS>.
- Package startup file <PK_STARTUP>.
- Package version command <PK_VERSION>.
- Package build command <PK_BUILD>.
- Startup dependency list <PK_STARTUP_DEPS>.
- Build dependency list <PK_BUILD_DEPS>.

The startup dependency list is the list of all packages that should be started up before <PK_NAME> is started up. Similarly, the build dependency list gives the list of all packages that must be built before <PK_NAME> is built. We use the symbol empty_list to represent an empty list in this context. Thus, the expression

<PK_BUILD_DEPS> = empty_list

means that no other packages need to be built before <PK_NAME> is built.

Similarly, not all packages have build commands. For such packages, we write <PK_BUILD> = empty_command.

The settings of the package parameters for dicemin are summarized as follows.

- Package name <PK_NAME> = dicemin.
- Package user directory: <PK_USER_DIR> = dice_user.
- Package download site: <PK_SITE> = Online Supplement [3].
- Package download file: <PK_FILE> = dicemin.tar.gz.
- Package definitions file: <PK_DEFS> = uxdefs_dicemin.
- Package startup file: <PK_STARTUP> = dicemin_startup.
- Package version command: <PK_VERSION> = dxmversion.
- Package build command: <PK_BUILD> = dxmbuild.
- Startup dependency list: <PK_STARTUP_DEPS> = empty_list.
- Build dependency list: <PK_BUILD_DEPS> = empty_list.

Package parameter settings for DICE and dicelang are listed in Section 8 and Section 9, respectively.

5.2 Step-by-Step Setup Process

The general setup procedure for packages that are based on DICE organizational conventions can be summarized by the following steps. The setup process for `dicemin`, which is described in detail in Section 4, can be viewed as a specific instance of this general setup process that is based on the `dicemin` settings for the package parameters listed in Section 5.1.

In the following sequence of steps, specific reference is made to the package parameters listed in Section 5.1. When referring to these parameters, the "package" qualifier may sometimes be omitted if it is understood from context (e.g., we may write "download file" in place of "package download file").

1. Download the desired package by downloading the associated download file `<PK_FILE>` from the download site `<PK_SITE>`. Then extract the package distribution directory from the `tar.gz` archive in which it is packaged.
2. Set up your `<PK_USER_DIR>` directory if the directory has not already been set up for another package that shares the same package user directory. For example, `dicemin` and DICE both use the same `<PK_USER_DIR>` directory (`dice_user`), so this step of setting up `<PK_USER_DIR>` can be skipped when setting up DICE (since `<PK_USER_DIR>` is created as part of the setup process for `dicemin`, which is typically installed before installing DICE).
3. Download the startup file `<PK_STARTUP>` from the download directory. Place the startup file in the `<PK_USER_DIR>/startup` directory (i.e., within a subdirectory called `startup` of `<PK_USER_DIR>`). Typically, this startup file is used "as is," and does not need any editing by the user.
4. From the package download site `<PK_SITE>`, download the package definitions (template) file `<PK_DEFS>` for essential "UX definitions". "UX" is a prefix used for environment variables that correspond to user-specific customizations for specific packages.
5. Place the package definitions file `<PK_DEFS>` in the `<PK_USER_DIR>/startup` directory.
6. Open the package definitions file `<PK_DEFS>` with a text editor. Set to an appropriate value the right hand side of each assignment statement at the bottom of the file. The meanings associated with these definitions should be provided as part of the package documentation or as comments in the `<PK_DEFS>` file (or both).

   Note that on some platforms, you may need to process the package definitions file and package startup files with the `dxrmcr` utility (part of `dicemin`) before they will execute properly. When setting up `dicemin`, `dxrmcr` is not available, and one may use an appropriately configured call to the UNIX `dos2unix` utility. Usage of `dos2unix` generally can vary across platforms; `dxrmcr` helps to provide a standard interface for the file conversion functionality that we need. Such processing just needs to be done once during initial setup.

6 General Startup Procedure

To use a package `<PK_NAME>` that is based on DICE organizational conventions, and to use any other packages that depend on `<PK_NAME>` in a given `Bash` session, one must first start up `<PK_NAME>` within that session. Starting up a package `<PK_NAME>` generally involves loading necessary environment settings so that you can use all of the features in `<PK_NAME>`.

6.1 Step-by-step Procedure

The general startup procedure for packages that are based on DICE organizational conventions can be summarized by the following steps.

NOTE: The instructions in this section (Section 6) do not apply to `dicemin`, since these instructions depend on certain utilities that are only available after `dicemin` is started up. To start up `dicemin`, one should refer to the instructions in Section 4.4.

To start up <PK_NAME>, based on the general package parameters listed in Section 5.1, follow these steps:

1. Make sure that you are operating within a `Bash` shell.
2. If the package startup dependency list <PK_STARTUP_DEPS> is non-empty, then make sure that you first start up all of the packages in <PK_STARTUP_DEPS>. The ordering of packages in <PK_STARTUP_DEPS> gives the order in which these packages should be started up (based on their respective startup dependency lists).
3. Go to the package user directory and invoke the package startup file from that directory. That is, run:

   ```
   cd <PK_USER_DIR>
   dxsource dx_load_package <PK_NAME> <PK_USER_DIR>
   ```

   As an example, the following command illustrates how DICE might be started up:

   ```
   dxsource dx_load_package dice $HOMES/me/user_directories/dice_user
   ```

6.2 Testing the Setup

As a basic test of the startup process and its preceding setup process, one can run the <PK_VERSION> command, which takes no arguments, from the enclosing `Bash` session after <PK_NAME> has been started up.

If the package has been properly set up and started up, the <PK_VERSION> command should execute and produce a (typically brief) message on standard output that gives the version number and other basic background information for the corresponding installation of <PK_NAME>.

Note that the <PK_VERSION> command is available as soon as the corresponding package has been set up — a package does not need to be built for its <PK_VERSION> command to be available. Thus, when installing a package, the <PK_VERSION> command can be used as a quick, basic test before one proceeds to the build phase, if there is one.

7 General Build Procedure

If <PK_BUILD> = `empty_command` for a given package <PK_NAME>, then this package does not have an associated package build command, and the instructions in this section (Section 7) can be skipped for <PK_NAME>.

Otherwise, to use all of the features available in <PK_NAME>, one needs to build certain executables from the corresponding source code.

NOTE: This step of building <PK_NAME> needs to be done only ONCE for a given version/update of <PK_NAME>. Typically, the build process is carried out just after a given version of <PK_NAME> is first setup, or just after one of the packages in the build dependency list has been rebuilt or otherwise updated.

To build <PK_NAME>, follow these steps:

1. If <PK_BUILD_DEPS> = empty_list, then skip this step. Otherwise, make sure that all of the packages in <PK_BUILD_DEPS> have been setup and built, following the instructions in Section 5.2 and Section 7, respectively, for those packages.
2. Make sure that you are operating within a Bash shell.
3. If <PK_NAME> has not already been started up within the current Bash shell, then start up <PK_NAME> (by following the instructions in Section 6).
4. Run the package build command <PK_BUILD> from the Bash command prompt.

Note that benign circular dependencies between packages may exist in the sense that a package P_1 may contain another package P_2 in its startup dependency list, while P_2 contains P_1 in its build dependency list. This is the case for example with DICE (as P_2) and dicelang (as P_1). Although such dependencies may be a bit confusing to work with at first, they do not cause problems if the packages are all started up and built using the proper sequences of steps. After you have ensured that P_1 is built, simply startup P_2 and P_1, and then, from the same Bash session, build P_2.

8 DICE Package Specification

The settings of the package parameters for DICE are summarized as follows.

- Package name: <PK_NAME> = dice.
- Package user directory: <PK_USER_DIR> = dice_user.
- Package download site: <PK_SITE> = Online Supplement [3].
- Package download file: <PK_FILE> = dice.tar.gz.
- Package definitions file: <PK_DEFS> = uxdefs_dice.
- Package startup file: <PK_STARTUP> = dice_startup.
- Package version command: <PK_VERSION> = dxversion.
- Package build command: <PK_BUILD> = dxbuild.
- Startup dependency list: <PK_STARTUP_DEPS> = {dicemin}.
- Build dependency list: <PK_BUILD_DEPS> = {dicemin, dicelang}.

Essential UX definitions for dice: the meanings of the environment variable settings that are involved in uxdefs_dice are as follows.

- UXARCH: This is used to represent the host platform on which your installation of DICE is being used. Available options are: lin (Linux), macos (Mac OS), sol (Solaris), and win (Windows).

- `UXDICE`: This should be set to the (UNIX/Cygwin-format) directory path of your DICE installation directory.

Based on the startup and build dependency lists, `dicemin` should always be started up before DICE is started up, and both `dicemin` and `dicelang` need to be built before DICE can be built.

9 `dicelang` Package Specification

The `dicelang` package, which can be viewed as a companion package of DICE, provides a collection of language-specific plug-ins that extend the features of DICE, and provide new features to facilitate efficient software project development, implementation management, and testing for selected programming languages. In contrast, the features in DICE emphasize generality, and applicability across different kinds of programming languages and development tools.

The settings of the package parameters for `dicelang` are summarized as follows.

- Package name: `<PK_NAME>` = `dicelang`.
- Package user directory: `<PK_USER_DIR>` = `dicelang_user`.
- Package download site: `<PK_SITE>` = Online Supplement [3].
- Package download file: `<PK_FILE>` = `dicelang.tar.gz`.
- Package definitions file: `<PK_DEFS>` = `uxdefs_dicelang`.
- Package startup file: `<PK_STARTUP>` = `dicelang_startup`.
- Package version command: `<PK_VERSION>` = `dlxversion`.
- Package build command: `<PK_BUILD>` = `dlxbuild`.
- Startup dependency list: `<PK_STARTUP_DEPS>` = {`dicemin`, DICE}.
- Build dependency list: `<PK_BUILD_DEPS>` = {`dicemin`}.

Essential UX definitions for `dicelang`: the meaning of the environment variable setting that is involved in `uxdefs_dicelang` is as follows.

- `UXDICELANG`: This should be set to the (UNIX/Cygwin-format) directory path of your `dicelang` installation directory.

Typically, the `dicelang_user` directory is created as a subdirectory of `dice_user`.

10 Combined Package Setup Sequence

Based on the detailed instructions described in the earlier sections, the following sequence of steps summaries the process of installing `dicemin`, DICE, and `dicelang`.

1. Set up the `dice_user` directory.
2. Set up `dicemin`.
3. Start up `dicemin`.
4. Build `dicemin`.
5. Set up DICE.
6. Set up the `dicelang_user` directory.
7. Set up `dicelang`.
8. Start up DICE.
9. Start up `dicelang`.
10. Build `dicelang`.
11. Build DICE.

11 `Bash` Startup File Integration

For a given package `<PK_NAME>`, the startup process can easily be included in a `Bash` startup file so that `<PK_NAME>` starts up automatically whenever the higher level startup file is invoked. Furthermore, sequences of startup dependencies can be adhered to systematically by ordering package startups appropriately in the higher level startup file.

For example, if the location for the DICE user directory is `~/dice_user`, one can add the code shown in Example 1 into a higher level `Bash` startup file so that the `dicemin`, DICE, and `dicelang` packages start up automatically whenever the higher level `Bash` startup file is invoked.

Example 1 Example code for starting up `dicemin`, DICE, and `dicelang` together from a higher level startup file.

```
cd ~/dice_user
source startup/dicemin_startup

cd ~/dice_user
dxsource dx_load_package dice ~/dice_user

cd ~/dice_user/dicelang_user
dxsource dx_load_package dicelang ~/dice_user/dicelang_user
```

The higher level startup file should be invoked using the `source` command in `Bash`. For example, if the code in Example 1 is stored in a script called `mystartup`, then one could use the following command to invoke the startup file.

```
source ./mystartup
```

This command should be executed from the directory that contains the script `mystartup`.

12 Setup from an Existing Installation

This section provides a modified version of the generalized setup instructions (Section 5.2) to enable use of a given package <PK_NAME> from a pre-existing installation. For example, such an installation may be a development version that is stored in a common repository or a networked installation that is maintained by a system administrator.

Below is a modified version of the setup instructions to work with a pre-existing installation.

- Follow Step 2 from Section 5.2.
- Follow Step 3 from Section 5.2.
- Follow Step 4 from Section 5.2.
- Follow Step 5 from Section 5.2.
- Follow Step 6 from Section 5.2. In the editing process, the variable (e.g., `UXDICE` for DICE or `UXDICEMIN` for `dicemin`) that "points to" the <PK_NAME> installation directory should be set to the location of the common installation directory.

13 Troubleshooting Guidelines

1. Directories and filenames with spaces. Except where otherwise specified, DICE generally does not work well with file or directory names and paths that contain spaces. This applies to arguments of utilities as well as DICE-related environment variable settings, and it often applies even if the arguments or settings are quoted. For example, a setting like the following one is likely to cause problems.

   ```
   UXDICE="/cygdrive/c/Documents and Settings/ann/My Documents/programs/dice"
   ```

   On the other hand, because the right hand side does not contain any spaces, the following setting is fine (even if the full path to the user's home directory includes directory names that contain spaces).

   ```
   UXDICE=~/utils/dice
   ```

   Note that an important exception to the "space problem" is the family of navigation-related utilities, which is described in [2]. In particular, `dlk` *can* be used to define labels for directories whose paths contain spaces. For example, the following sequence will work as long as the targets of the `cd` commands are valid.

   ```
   cd "/cygdrive/c/Documents and Settings/ann/My Documents"
   dlk mydocs
   cd ~/somewhere_else
   g mydocs
   ```

2. Windows-style paths. When using DICE under Windows, it is important to keep in mind that DICE generally requires UNIX-style paths. Such paths should not contain Windows drive specifiers such as `C:` or `D:`, nor should they contain the backward slash character (\) as a directory separator.

   For example, the following are likely to cause problems.

   ```
   UXDICE=C:/smith/downloaded/dice
   UXDICE=C:\\smith\\downloaded\\dice
   ```

On the other hand, the following forms are acceptable.

```
UXDICE=/cygdrive/c/smith/downloaded/dice
UXDICE=~/smith/downloaded/dice
```

3. `DXVERBOSE`. A useful diagnostic feature to use when setting up DICE and DICE plug-ins is the `DXVERBOSE` environment variable. If this variable is set before starting up DICE to any non-empty-string value (e.g., with `DXVERBOSE=on`), then diagnostic comments will be displayed as DICE is started up. This is usually not needed, but can be useful to help diagnose or report a problem if DICE or a DICE plug-in fails to start up properly.

4. `DXVERBOSE` for DICE utilities. Some DICE utilities produce diagnostic output that is enabled by the `DXVERBOSE` environment variable, as described above. Thus, the `DXVERBOSE` variable can also be useful in diagnosing configuration or usage problems related to DICE that are not associated with the setup of DICE or the DICE startup process.

5. Exiting during startup. Another point that is useful to keep in mind when troubleshooting DICE setup issues is that the enclosing `Bash` shell may exit if DICE or one of its plug-in packages fail to startup. If this happens, then enable verbose output, and redirect standard output to a file in your startup command.

   For example, to apply this method when starting up DICE, one can use the following sequence of commands.

   ```
   DXVERBOSE=on
   source startup/dice_startup > ~/startup-log.txt
   ```

   Then one can view the `~/startup-log.txt` file for a transcript of the failed startup session.

   Similarly, if the DICE user directory is `~/dice_user`, then to apply this troubleshooting approach when starting up `dicelang`, one can use the following command (from any directory).

   ```
   DXVERBOSE=on
   dxsource dx_load_package dicelang ~/dice_user > ~/startup-log.txt
   ```

14 Acknowledgments

This work is sponsored in part by the U. S. National Science Foundation under grant NSF-ECCS0823989, the Laboratory for Telecommunication Science, and the US Air Force Research Laboratory.

We are grateful also to the following people who have made valuable contributions to DICE, and earlier software components that have evolved into parts of DICE and `dicelang`: Bishnupriya Bhattacharya, Nitin Chandrachoodan, Soujanya Kedilaya, and Robert Ricketts.

References

[1] "Maryland DSPCAD Research Group Website," `http://www.ece.umd.edu/DSPCAD/home/dspcad.htm`.

[2] S. S. Bhattacharyya, W. Plishker, C. Shen, N. Sane, and G. Zaki, "The DSPCAD integrative command line environment: Introduction to DICE version 1.1," Tech. Rep. UMIACS-TR-2011-10, Institute for Advanced Computer Studies, University of Maryland at College Park, 2011.

[3] "DICE user's guide online supplement," `http://www.ece.umd.edu/DSPCAD/projects/dice/guide/supplement.htm`.