Up until this point we have been creating ad-hoc queries. Ad hoc queries are not saved when you exit a DVDev session, so to save them you have to back them up externally (on your own network/computer). Alternatively, you could create an executable program and save them in your Cerner environment.

By using executable programs, you can perform the following tasks:

* Make the execution process more efficient by eliminating the need to include the commands each time it is executed.
* Create PROMPTS to pass parameters into the program during execution.
* Run the program from the following locations:
  + *Discern Explorer* command line
  + *Discern Explorer* Menu (eliminated in 2018)
  + DA2
  + Batch
  + Operations
  + Discern Expert (Rules)

**6.1 Overview**

The process of creating an executable program requires the understanding of two different terms. *Source code* refers to the text or syntax in your program. The file that you create using DVDev is considered your source code and follows the rules and syntax defined for CCL. DVDev takes this source code and creates an *object* that is in a format the computer can understand.

This object is stored in a file referred to as the *object library* or the *dic.dat* (Discern Explorer Data Dictionary)*.* The object library is the repository of executable programs.

When you create a source code file composed of CCL commands and then include that source code file, you have created and added an object to the object library. An object, then, is nothing more than an executable program that is stored in "computer language" which can be run without re-compiling the file.

When you need to make modifications to an executable program (the object), you make the modifications to the source code, not to the object itself. You must then re-include the source code file, replacing the old object with the new object that reflects the changes made to the source code file.

Changes made to the source code file are not reflected in the executable version (the object), until you re-include the source. In the following sections, the DROP PROGRAM and CREATE PROGRAM commands create and modify an object.

**6.2 Creating an Executable Program**

In your non-production environment, open up the application *DiscernVisualDeveloper.exe*.

Machine generated alternative text:
File Edit 
Code View 
Code View 
Output 
Discern Visual Developer 
View Build Tools Reports Window Help 
NUM 
Request/RepIy Tables/FieIds 
Macros 
MYO 8134316 Lno, colo 

Once in the application, click on the white paper icon *under File* and then select *New (or just type Ctrl + N)*. Select *Program* under the File Type.

The Program Name is the name of the object. When you fill out the Program Name, the File Name automatically populates with the same name followed by PRG file extension. This is the file extension for every executable CCL program. You can name the File Name whatever you want, but it's good practice to name it the same as the Program Name.

Use your initials followed by test as the Program Name.

Machine generated alternative text:
Fie Type 
Prompt am 
Ad Hoc Query 
am Wizard 
Free Form Label 
Discern Expert T ernplate 
program Name 
ym_teÅ 
Fie N Moe 
Logical 'Path Save Location 
CCU-ISERDIR: 
Node: 
( def") 
n use Header 

When creating an executable program, a file is created and stored on the back-end. You can control the location this file is stored by changing the Logical/Path as well as the node. For now, leave the Node as default. The first time you create a file there likely won't be any location listed under Logical/Path, so type in CCLUSERDIR: (you need the colon), click OK, and save the file when the next prompt appears.

If you happen to enter a file directory that doesn't exist when you try and save the program you will receive the following error:

Machine generated alternative text:
ccluserdird:ym test.prg failed to save. 

At this point, we have created a file for our source code, but we haven't actually created an object. If you have back-end access you can navigate to your own CCLUSERDIR directory and do an ls with your own filename. You should find it there.

C:\Users\b134316\AppData\Local\Temp\msohtmlclip1\02\clip_image004.png

Now, go to Reports -> CCLPROT and type your Program Name in the Object Name field and execute it. You should see no entry for you object.

Machine generated alternative text:
Output to FWPtinteu'MlNE 
O blect Type 
Object Name 
Inch& Souce Name. 
Prompt CCLPROT 
MINE 
program 

CCLPROT will be discussed in greater depth in Chapter 8 Finding the Right Data. For now, all you need to understand is that it's a report that you can run that determines whether an object exists or not.

Saving a file does not include an object in the object library. You must actually compile the file to do that.

Note: Saving updates the source code file stored on the backend while compiling creates an object in the object library.

Since Cerner uses the object library when running a program, you can make an infinite number of changes to the source code without actually updating the object. Only when you compile the source code will the changes take effect.

You should see the standard template for all new programs. Programs begin with the drop program / create program keywords and end with the end / go keywords. The drop program / create program add and remove the object from the object library.

Machine generated alternative text:
drop program rm test go 
create program y-m test 
DVDev 
DVDev 
DVDev 
DECLARED 
DECLARED 
SUBROUTINES 
VARIABLES 
Start 
Coding 
Goes 
Code 
Here 
Your 
DVDev 
end 
DEFINED 
SUBROUTINES 

Cerner won't let you compile a program until there is code, so let's do that now. Replace all of the comments with the following line of code:

*call echo("Hello from inside my program")*

Echo is a function that takes a single argument. Its purpose is to print text to a listing file, which is discussed in greater depth later in the chapter.

Compile the program using the icon that looks like a blue arrow pointing on a brick wall and save the program when prompted.

Machine generated alternative text:
Include/CompiIe current view 

The output window on the bottom of the screen will let you know whether compiling the file was successful or not.

Machine generated alternative text:
Output 
lldrop program ym test go 
%CCL-W-42-EKS TEST)Could not destroy program NM TESTII since not found in object lib. 
%CCL-I-1 9-EKS denied on object NM TEST) with type (PI due to (rdbmsl. 
CCL compile with - 0 error(s), 1 warning(s), 1 informational 

The first warning is simply indicating that the drop program command couldn't be run because this is the very first time we are including our program. If you compile a second time, you no longer see this warning. The next line is informational and it can be ignored - you may or may not see this in your environment, but it's an issue that Cerner is working on fixing.

If you re-run the CCLPROT report, you should now see your object in the object library!

Machine generated alternative text:
YM TEST 
GRCUP TYPE 
(E) xe cute 
P 3134316 
SIZE 
(S) elect (R) ead 
9002.0. o 
CC VER DÄ2E 
(0 TO 10 PROTECTION) 
(W) rite (D) elete (I)nsert (U)pdate 
2 Reg 
App—a, o, o 
test.prg 

**6.3 Program Security**

When a user creates an executable program, the object is stored under their user group in the object library. When a user runs an executable program, CCL looks for the object in that user's group in the object library. In other words, the user group controls access to the program.

It's important to understand how this works. If you run the CCLPROT on your program, you should see a numeric value for group type. In the screen shot below, the group type is 0. Since I compiled the program that means I have group 0 access.

Machine generated alternative text:
YM TEST 
GRCUP TYPE 
(E) xe cute 
P 3134316 
SIZE 
(S) elect (R) ead 
9002.0. o 
CC VER DÄ2E 
(0 TO 10 PROTECTION) 
(W) rite (D) elete (I)nsert (U)pdate 
2 Reg 
App—a, o, o 
test.prg 

The group type defines a users' security level. A group 1 or Database Administrator (DBA) has administrative privileges and can create and execute programs and is the default group for all users. Groups 2-99 have no privileges in CCL. A DBA must grant these users access to specific programs. These groups are used to restrict and define the privileges a user has to CCL programs. Many scripts must be included as group 0 to properly work with rules, interface scripts, ops jobs and other processes.

The aforementioned security measures were created to allow multiple versions of the same program to be created that are run depending on user access. Generally speaking, this is not a good idea. Plus, it's a support nightmare.

Machine generated alternative text:
YM TEST 
-cap pa rnfnp 
source—ccLIJSERDIR . 
• ym_te st . prg 
GRCUP TYPE 
(E) xe cute 
P 3134316 
P 3134316 
SIZE 
(S) elect 
(R) ead 
9002. . 
9002. . 
CC VER DÄ2E 
(0 TO 10 PROTECTION) 
(W) rite (D) elete (I)nsert (U)pdate 
Reg 
Reg 15un_TL1g 
App—a, o, o 

If the same object is included with a different group, you will receive a duplicate warning. It's best to only have a group 0 or a group 1 report.

**Note**

CCL security does not control security to RDB tables. You must use your RDB sign-on to control access to specific tables.

**6.4 Running an Executable**

There are many ways that scripts can be executed: DVDev, prompts, command line, rules, ops, etc. We will be executing almost every program through either DVDev or prompts.

There are two ways to run a program in DVDev. The first is that you call your program *within your program*. This is what it looks like:

Machine generated alternative text:
rop program rm test go 
create program rm test 
call echo ("Hello from inside my program") 
end 
execute 
y-m test 
go 

When the program is compiled, it's executed *at the same time*.

This method doesn't work in every scenario, however. There will be many times when you will be executing code you do not have access to modifying (Cerner standard code) or you may be troubleshooting a script in a production environment, so you don't want to make any changes to it. In these instances, there is a more preferred method to executing a program.

Ensure that your program is compiled with the single call echo statement.

Machine generated alternative text:
drop program rm test go 
create program y-m test 
call echo ("Hello from inside my program") 
end 

Type CTRL + N and then Enter to create a new blank tab. In the blank tab, type the following command:

execute <program\_name> go

Machine generated alternative text:
CCLUSERDIR:ym_test.prg 
B134316 DVD4* 
execute rm test g 

Compile the program and when prompted to save, click No. As soon as you hit No you are actually executing your program.

Machine generated alternative text:
Save changes to 8134316 DVD4 

It doesn't really look like much though, does it? Aside from the output at the bottom of the screen, nothing happens!

Cerner has a log called a Listing that, well, lists all of the output from the program that was run. Type CTRL + L or click on the listing icon right next to the compile icon and a second window should open. It's important to note that when you run the listing, you must be on the tab with the *execute <program\_name> go* statement or it will not work.

Machine generated alternative text:
Listing 

The listing shows the echo statement.

Machine generated alternative text:
I) execute rm test 
go 
ell o from inside 
my program 
1) 
190716: OES649 
0 00 
0 00 
3134316 DVDI 
Cost 
cpu 
Ela 
Dio 
COMORO 
P2R0 

It's vital to master this method of executing programs because it's the fundamental basis for executing programs. Virtually every single program going forward will be executed in this manner and any advanced concepts like mimicking front-end applications and trace level logging build on this premise. Take your time to go through the exercises to ensure you have a sound understanding of how executing programs from the front-end work and how source code and objects differ.

**Exercise 6.1 Version Control**

1. Create a new program and save it as version 2 (if you haven't already created version 1 do that now). My first version was called ym\_test, so version two is ym\_test2.
2. If you have back-end access, navigate to the location in which you saved it just to prove to yourself that you created a .prg. You should see two files, one for each version.
3. Run CCLPROT from Reports -> CCLPROT making sure Include Source Name is set to Yes. Your object name should exist and the source should *still be from the original file*. Remember: you haven't compiled yet.

Machine generated alternative text:
YM TEST 
GRCUP TYPE 
(E) xe cute 
P 3134316 
SIZE 
(S) elect 
9002.0. o 
CC VER DÄ2E 
(0 TO 10 PROTECTION) 
(W) rite (D) elete (I)nsert (U)pdate 
2 Reg 
App—a, o, o 
test.prg 

1. In your new, version 2 program (ym\_test2) rename ym\_test2 to ym\_test after drop program and create program. The name listed here is the name of the object that will be included in the object library.
2. Add the following statement to version 2:

*call echo("Hello from inside my program version 2")*

Your program should look something like this:

Machine generated alternative text:
drop program rm test go 
create program y-m test 
call echo ("Hello from inside my program version 2" 
end 

1. Open a new window and execute your object using the method shown in section 6.4. When you run the listing you should see the echo statement from your original file ("Hello from inside my program").
2. Close the listing and navigate back to your un-saved, version 2 file. Compile it.
3. Rerun CCLPROT. You should still see your object included, but the source should now be from your version 2 file.

Machine generated alternative text:
YM TEST 
GRCUP TYPE 
(E) xe cute 
P 3134316 
SIZE 
(S) elect 
(R ) e ad 
(W) rite (D) elete (I)nsert (U)pdate 
9002.0. o 
2 Reg 
App—a, o, o 
CC VER DÄ2E 
(0 TO 10 PROTECTION) 
test2.prg 

1. Open a new window and execute your object just like step 6. When you run the listing you should see the echo statement from your version 2 file!

This is how version control generally works in Cerner (however, in 2018.01 a package is being released for Git integration). In this example, you created a simple program that echo'd out a simple statement. You then over-wrote the object from a new file with a different echo statement. Versioning in this manner is good practice because it's extremely easy to revert a change in the event of an error. If version 2 has an issue, all you have to do is open up version 1 and re-compile.

**Exercise 6.2 Passing in Prompts**

1. Open the program you created in exercise 6.1 and add the following call echo statements:

call echo($1)

call echo($2)

call echo($3)

1. Open a new window and execute your program like before, but this time add three arguments before executing. Your code should look something like the code below. Each argument is separated by a comma.

*execute ym\_test*

*"some programs accept arguments"*

*, "arguments can be passed in like this"*

*, "or in ways we will learn later"*

*go*

1. Your listing should echo out each of the arguments you passed in.

Machine generated alternative text:
I) execute rm test 
2) "some programs accept arguments" 
"arguments can be passed in like this" 
4), "or in ways we will learn later" 
ell o from inside my program version 2 
some programs accept arguments 
arguments can be passed in like this 
or in ways we will learn later 

When you execute a program with arguments, Cerner will automatically assign them to variables ($1, $2, $3, etc.) unless variables are specified with a prompt program (we will learn about this later). More often than not, you will never use the Cerner assigned variables, the programs you will be troubleshooting will likely have a prompt program associated.

However, regardless of whether the prompt program exists or not, this is how you execute an executable that has a prompt program.

**Exercise 6.4 Security**

1. Open version 2 of your program and append :group1 to the end of your object name. Mine looks like this:

*drop program ym\_test:group1*

*create program ym\_test:group1*

1. Compile the program and run CCLPROT. If you have group 0 access, you should see two records listed. If you only see one (and it's group 1) then compile your program appending :group2 instead of :group1

Machine generated alternative text:
YM TEST 
-cap pa rnfnp 
source—ccLIJSERDIR . 
• ym_ 
GRCUP TYPE 
(E) xe cute 
P 3134316 
P 3134316 
prg 
SIZE 
(S) elect 
CC VER DÄ2E 
(0 TO 10 PROTECTION) 
(W) rite (D) elete (I)nsert (U)pdate 
Reg 
Reg 
App—a, o, o 
9002. . 
9002. . 

1. Copy the drop program command from your program into a new window and compile it. If you re-run CCLPROT you should only see one version again.

*drop program ym\_test:group1 go*

By now you should understand the different between a source file and an object in an object library. You should be able to create a program, compile it, execute it, and view a listing of its output.

You may be wondering how to integrate a select statement into a program that you can keep running and that is totally understandable. Unfortunately, it requires understanding several concepts like prompts and several new control options. If that is your goal then I recommend jumping to the Prompts chapter. Otherwise, we will be moving into control flow and ways to store data temporarily.