CDEV Script Service

A Tool for Using Interactive Scripts from CDEV

Walt Akers

Version 1.0 - February 11, 1997

TJNAF - Thomas Jefferson National Accelerator Facility

DOCUMENT DATE: Table of contents generated: February 13, 1997 2:57 pm

SURA/CEBAF: The Southeastern Universities Research Association (SURA) operates the Continuous Electron Beam Accelerator Facility (CEBAF) for the United States Department of Energy under contract DE-AC05-84ER40150.

Table of Contents

List of Figures

1. Script Service Fundamentals

Overview of the Script Service

The CDEV Script Service is designed to allow a a CDEV application to execute an external script to process a message. The output that is produced by the executable will be returned to the caller in the form of a cdevData. The CDEV Script Service meets the following requirements.

Features of the Script Service

- The filename of the script is associated with a CDEV device/message combination by using the “filename” field of the service data in the CDEV DDL file. Syntax is described later in this document.
- A new process will be spawned for the script each time it is executed. Its standard output (stdout) file descriptor will be redirected to return data to the main CDEV application, allowing the script to return results to the parent process.
- The external executable file can be any form of user shell, batch file or binary application that is supported on the host platform.
- The Script Service supports all of CDEV’s send mechanisms; “send”, “sendNoBlock” and “sendCallback”. Because of limits imposed on the number of active processes, the user is advised to use the synchronous “send” method whenever practical.
- When called by CDEV, the script will receive three arguments; the name of the CDEV device associated with the request, the message string that was submitted to the device, and a string that describes the contents of the outbound cdevData object.
- The reply that is written to the standard output by the script must have a specific format as described later in this document. A script may return more than one reply packet to the caller.
- The script notifies CDEV that it has written its last reply packet by writing “done” alone on a line to the standard output.
- If the script is to return multiple results, it should write “end” alone on a line to the standard output between each packet, and write “done” alone on a line following the last packet.
- If the script returns a value in the “status” tag of its reply packet, this will be provided to the caller as the completion code of the message. Traditionally a status of 0 indicates success, and all other values indicated an error occurred.
- Because the callback mechanism of the Script Service is triggered by output from the script, the script developer should at least write “done” to the standard output even if the application returns no output.

2. Building the Script Service

Location of Files

The source code for the Script Service is provided with the CDEV distribution starting with version 1.5. The source code is located in the directory $CDEV/extensions/ScriptService. The following steps should be taken to build the ScriptService.

Steps to Building Service

1. Obtain and install the CDEV distribution - Version 1.5 or higher.

2. Follow the installation steps to build CDEV on your system. This includes setting all of the environment variables required by CDEV.

3. Change directories to *$CDEV/extensions/ScriptService/src*.

4. There is a collection of platform specific makefiles in the directory *$CDEV/extensions/cdevGenericServer/include/makeinclude*. Link the makefile that is appropriate to your architecture to the file Makefile.OS.

   For example, to link to the makefile for HP-UX, you would type the following command.

   ln -s Makefile.hpux Makefile.OS

5. Type make to compile and link the Script Service shared object. Note that you must be using GNU make in order to build this distribution.

Upon completion the Makefile will generate the shared object for the Script Service and will install it into your CDEVSHOBJ directory. At this point you are ready to use the Script Service.

3. Structure of Data Provided to the Script

General Form of Data String

When a developer makes a CDEV send call, he may provide outbound data in the form of a cdevData object. Before calling the script, this data is converted into a string and provided as the third argument to the application (the first argument is the device name and the second is the message). The structure of the outbound data string is of the following form.

Figure 1: General form of string representation of cdevData

```
tag1=”string value”
tag2=1.00
```

Representation of Scalar Data

The above example shows two data entries of the form *tag=value*. Each entry is separated by a carriage return. The first entry has a tag name of *‘tag1’* and a value of *‘string value’*. You will note that *‘string value’* is enclosed in double quotes in the example. This is used to notify the script that the value is a non-numeric character string. In the second data entry the value *’1.0’* is not enclosed in quotes, indicating that the value is a number.

The following example illustrates how scalar values stored in a cdevData object would look when they are passed to the script.

Figure 2: String representation of scalar data in a cdevData object

```
Data stored in cdevData object:
      Tag:value
      Data Type:character string
      Value:Test

      Tag:status
      Data Type:integer
      Value:0

      Tag:controlHigh
      Data Type:double
      Value:1.001

String representation provided to script:
      value=”Test”\n
      status=0\n
      controlHigh=1.001\n
```

Representation of Array Data

If array data is stored in the outbound cdevData object, it too can be provided to the script. The data is structured in the standard C format where opening and closing braces mark the beginning and end (respectively) of each array bound. The following example illustrates how various arrays would be represented as strings.

Figure 3: String representation of array data

```
Data stored in cdevData object:
      Tag:value
      Data Type:character string [3]
      Value:"value1" "value2" "value3"

String representation provided to script:
      value={"value1","value2","value3"}\n
```

```
Data stored in cdevData object:
      Tag:value
      Data Type:double [3]
      Value:1.0, 2.0, 3.01

String representation provided to script:
      value={1,2,3.01}\n
```

```
Data stored in cdevData object:
      Tag:value
      Data Type:double [3][2]
      Value:1.0, 2.0,   3.0, 4.0,    5.0, 6.0

String representation provided to script:
      value={{1,2},{3,4},{5,6}}\n
```

4. Structure of Data Returned from the Script

Returning Data to the Calling CDEV Application

Before CDEV starts the script, it creates a pipe that is used to pass data between the script and the CDEV application. One end of this pipe is maintained by the application and the other end of the pipe is provided to the script as its standard output. This means that anytime the script uses the "echo" command or the "printf" command, the data that it output's will be transmitted through the pipe to the parent application.

The script is responsible for formatting the data into a collection of character strings that can be recognized by the CDEV application.

General Form of String Data

In general, the data that is returned from a script is of the same form as described in the preceding section, "*Structure of Data Provided to the Script*". The most notable difference is that the script must terminate the list by providing a character string that indicates that the group of data items is complete; the string "*end*" or "*done*" alone on a line.

Returning a Single Result Packet

If the script is going to return exactly one result packet (*a group of tags and values that will be used to populate the result cdevData*), then it should write the string "*done*" on a line by itself immediately following the output data. This will notify the Script Service that no more data will be returned from the script.

The following example shows how a single result will be returned from a script. Note that the "status" tag contains the completion code that will be returned to the CDEV application.

Figure 4: Returning a single result packet to the CDEV application

```
#\! /bin/csh -f

echo value=\"Test\"
echo status=0
echo controlLow=1.5
echo controlHigh=25.1
echo done
```

Returning Multiple Result Packets

Returning multiple results from a single script execution is not generally advised. However, If the script is going to return more than one result packet, then each interim packet should be terminated by the "*end*" keyword on a line by itself. The "*done*" keyword should follow the last entry.

The following example shows how two results will be returned from a script.

Figure 5: Returning a multiple result packets to the CDEV application

```
#\! /bin/csh -f

echo value=\"Start Result 1\"
echo status=0
echo end
echo value=\"Start Result 2\"
echo status=-1
echo done
```

Special Considerations

If the script that is being executed will not return a result, then the developer should still write "*done*" to the standard output in order to notify the CDEV application that it should no longer wait for output. If the script is not returning a result, it may write the terminating *"done"* at the beginning of the script or at the end. Because synchronous calls will block until a reply is received, the developer should return a result to the CDEV application as quickly as possible.

If the developer never writes anything to the standard output, then the user specified CDEV callback will never be executed... therefore, it is crucial to notify the CDEV application when the script has completed.

5. DDL Entries for the Script Service

Setting Up the DDL

The following example illustrates how to add a service entry to identify the Script Service and how to use the service data to specify the filename that the Script Service will execute to service a message.

Figure 6: Simple DDL file for the Script Service

```
/*
 * This is the service definition, it tells CDEV that there is
 * a service named Script and that it will use the cdevData
 * tag 'filename' to read configuration information from the
 * DDL file entries.
 */
service Script
   {
   tags {filename}
   }

/*
 * This class definition identifies the class scriptClass which
 * supports the "get and "set" commands on attributes "attrib0"
 * and "attrib1".  Note that attrib0 will call script /bin/ouch
 * and attrib1 will call script /bin/wammo.
 */
class scriptClass
   {
     verbs {get, set}
   attributes
       {
       attrib0 Script {filename = /bin/ouch};
        attrib1 Script {filename = /bin/wammo};
       }
}

/*
 * Finally, the CDEV DDL identifies two devices that are of type
 * scriptClass.  Any time one of the supported messages is passed
 * to one of these devices it will activate the corresponding
 * script.
 */
scriptClass : device0, device1;
```