Section M10 J-DSP Scripts

1. J-DSP Script basics

J-DSP is fitted with an interpreter designed to decode parameters contained in a simple Hypertext Markup Language (HTML) file, which when loaded through a browser, invokes the J-DSP editor. The editor in turn interprets the parameters and loads a J-DSP flowgram as described by these parameters. The parameters contained in the HTML file are written in JavaScript and are actually components of a Java™ applet referred here as a J-DSP script. This J-DSP Editor capability has been designed in order to allow instructors save their own J-DSP simulation examples on the Internet, easily integrating interactive content in classroom web sites.

In addition to this introduction, this manual consists of two more main sections that further discuss J-DSP scripts. Section 2 of this manual instructs the user on how to create J-DSP scripts automatically, while section 3 is an elaborate description of how to manually prepare the script code.

Please note that all scripts must be saved in the same directory the J-DSP editor's class files are located in.

2. Generating scripts automatically.

While trivial to prepare manually, the J-DSP Editor has been provided with the ability to automatically generate J-DSP scripts. A user simply needs to create the desired flowgram using the familiar drag and drop procedure of the editor. Then, by selecting "*File"* and "*Export as Script"*, the user obtains the script, ready to use in an HTML file. This automatically generated script also includes all the blocks parameters, exactly as they where defined when the simulation's flowgram was saved, providing full control over a saved simulation. Section 2.1 contains step by step instructions on how to create your own J-DSP scripts.

2.1. Scripts in a flash.

A few steps is all that is needed to get a J-DSP script and save it over the Internet for others to use.

Step1: Start the J-DSP Editor and create a flowgram as desired. Don't forget to set the desired parameters to all the parts.

Step 2: From the main menu, select "*File"* and then "*Export as Script"* as shown in figure 1. A new window appears that contains the code describing the simulation, as shown in figure 2. This will be referred to as the J-DSP script window. Note that by default, the window presents only the applet code which can be placed inside the body of an existing HTML file. If the entire HTML file is needed, select "*Applet in HTML code"* from the drop down box of the script window.

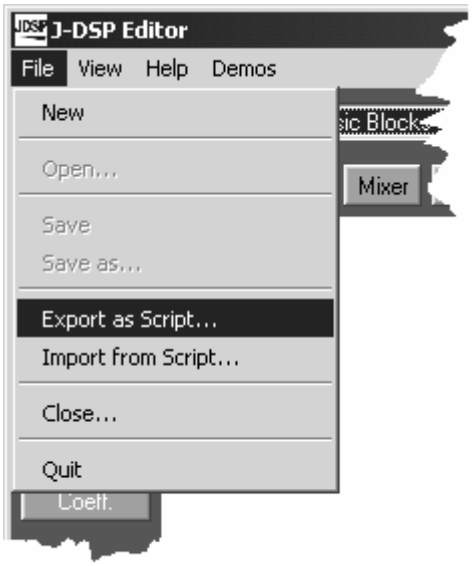


Figure 1: Selecting *File* and then *Export as Script*

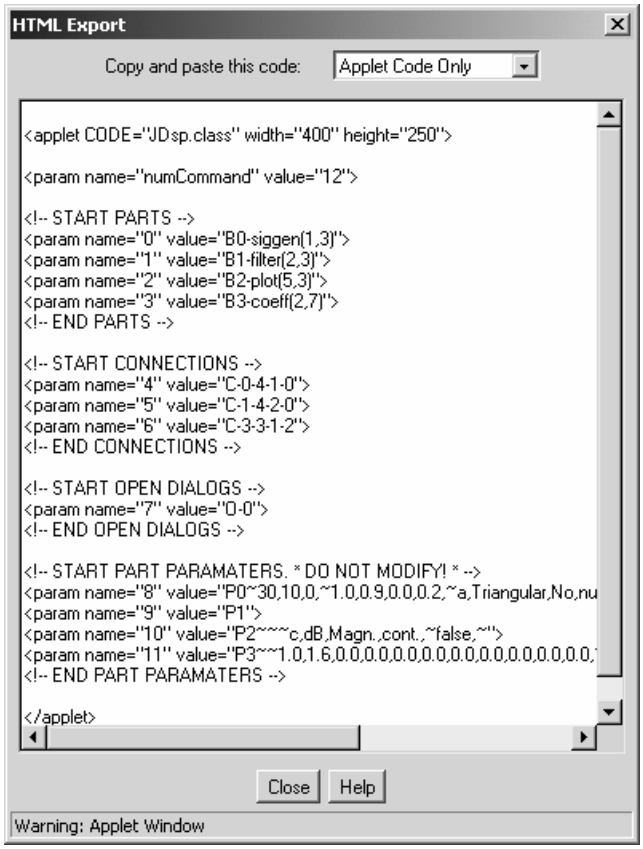


Figure 2: J-DSP script window

Step 3: Using the mouse, select the code from the window and press *Ctrl-C* to copy it into the clipboard. Some users may use a right click and then *Copy*, depending on the Java version used on their computer. Non-Windows™ users should be able to follow a similar procedure. ***Note***: If you are not able to use Ctrl-C to copy, please install the latest java virtual machine from Sun

found at www.java.sun/getjava/. Instructions are provided in the troubleshooting section of our web site located at http://jdsp.asu.edu.

Step 4: If you wish to place the applet in an existing HTML file, simply paste the script code at the location where you desire the J-DSP Editor applet to appear. If you do not have an HTML file, copy the entire HTML code into a text editor and save the file with an .html or .htm extension. Steps 4.1 and 4.2 provide some extra details and can be skipped if you feel comfortable with what was just mentioned.

Step 4.1: Saving into an existing HTML file:

a. Use an HTML editor or any text editor capable of reading ASCII files to load the *filename.html* or *filename.htm* file you wish to use for the script. If your HTML editor is a "what you see is what you get" (WYSIWYG) editor, make sure to select to edit the HTML code itself.
b. Place the cursor where desired and paste the applet code using *Ctrl-V* or right click and *Paste* as shown in figure 3.
c. Save the file and then load it in a browser: ***Note***: Do not attempt to view the HTML file through the HTML editor's internal browser, as it might not be a fully functional browser and therefore not capable of displaying active content like Java™ applets.

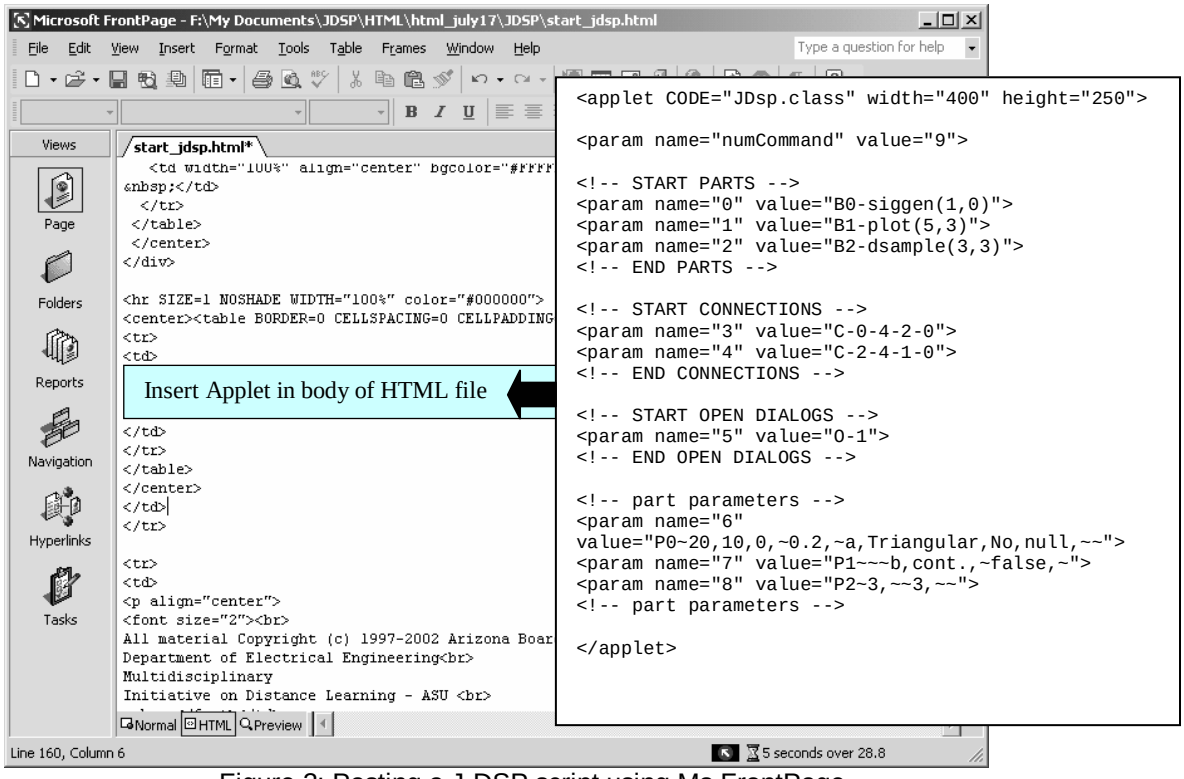


Figure 3: Pasting a J-DSP script using Ms FrontPage

Step 4.2: Saving into a new HTML file:

a. Use any text/HTML editor to create a new file.
b. Copy and paste the entire HTML code from the script window into the text/HTML editor.
c. Save the file with an extension of .htm or .html, so that any browser can recognize it. For example, you can name the file: myjdsp.html.

Important note: In every case, make sure that the HTML file containing the script is saved where the J-DSP editor class files are saved, otherwise the script will not run.

Step 5: After saving the HTML file in the same directory as the J-DSP script files, load the file in a browser and press the [Start] button. This should start the J-DSP Editor and load the saved simulation. Make sure you place a link to this HTML file from your web page so that others can have access to it.

You have now created a J-DSP script. If you are interested to learn more on how J-DSP scripts work, you can read further to section 3, otherwise you can safely stop here.

3. Generating scripts manually – Writing the code.

Typically only a few lines of code are necessary to set up and execute a simulation. The following sub-sections describe how to do so, through a simple example.

3.1. The Basics.

An example of html code that establishes and runs a simple J-DSP simulation is listed below:

```
0   <applet CODE="JDsp.class" width="400" height="250">
    
1   <param name="numCommand" value="3">
    
2   <param name="0" value="B0-siggen(3,1)">
3   <param name="1" value="B1-plot(5,1)">
4   <param name="2" value="C-0-4-1-0">
    
5   </applet>
```

Note that the J-DSP scripts are placed between the html applet tags, namely, <APPLET> which marks the beginning of the J-DSP applet, and </APPLET> that marks the end of this applet. The applet tag, <APPLET>, also includes the following information: it instructs the browser to load the applet by specifying its sub-class name, width (400 pixels), and height (250 pixels). In our case, the user should always type the following:

```
<APPLET CODE="JDsp.class" WIDTH="400" HEIGHT="250">
        J-DSP scripts go here
</APPLET>
```

To create a J-DSP flowgram with scripts, you have to specify a set of parameters that establish the blocks and link them to form a flowgram. These parameters are not associated with the DSP simulations but instead they are merely code to establish the simulation. These scripts activate the J-DSP applet containing the [Start] button. By pressing this button, the programmed flowgram will appear.

To create J-DSP scripts, an HTML parameter tag, known as the <PARAM> tag is used. The use of this tag is described below. The general format for a <PARAM> tag is:

<PARAM NAME="*parameter1Name"* VALUE="*a Value"*>

The first line of the J-DSP script is a <PARAM> tag, which specifies the total number of J-DSP script lines. The parameter name is "numCommand", and its value is the number of <PARAM>

tags that will be used in this particular applet, excluding itself. This will become clearer as the <PARAM> tag description continues. The format for the first line of the script is:

<PARAM NAME="numCommand" VALUE="*number of param tags below*">

For the general <PARAM> tags that follow, each tag's PARAM NAME is given a unique number. This number increases sequentially starting from 0. For example,

<PARAM NAME="0" VALUE="*a Value*">
<PARAM NAME="1" VALUE="*another Value*">

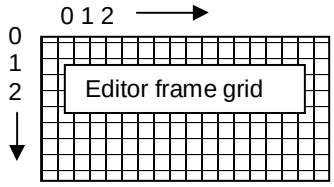
The PARAM NAME, VALUE etc keywords are case insensitive, so it is not necessary to use capital letters.

In the VALUE part of the <PARAM> tag, the parameters **B**, **C**, **O**, **P** and ***** are used to specify different instructions. For example, **B** is used to instruct the editor to create a new part (block). These parameters are further explained in the following sub-sections.

3.2. Creating parts.

One important instruction that can be given to the J-DSP editor is to create a new part, or block as parts are often referred to. Therefore, **B** represents a new block, and is followed by a number *i*, which specifies that this is the i^{th} block of the flowgram. This number is also used to refer to the part when performing other tasks with scripts, like opening its dialog box. The block name follows, with the coordinates of the editor frame in parentheses.

For example, **B**0-siggen(3,1) means that the first block of the flowgram is a **SigGen** block and is to be placed in the coordinates (3,1). The editor frame grid is shown below. Each box is approximately 50x70 pixels.



Here is the <PARAM> tag for this case:

<param name="0" value="**B**0-siggen(3,1)">

Now we are ready to write a short and simple program to establish a block in the editor frame:

0	<applet CODE="JDsp.class" width="400" height="250">
1	<param name="numCommand" value="1">
2	<param name="0" value="**B**0-siggen(3,1)">
3	</applet>

Observe that the first <PARAM> tag in line 1 with "numCommand" as PARAM NAME, has VALUE = "1" because there is only one general <PARAM> tag in this program. This is the <PARAM> tag in line 2. Line 2 establishes a ***SigGen*** block at the location given by the coordinates (3,1).

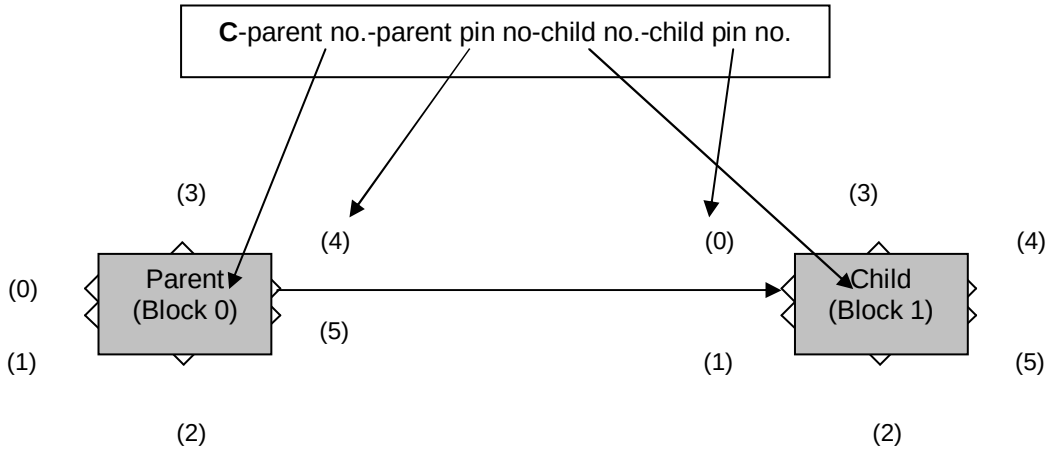
If we include the applet above in an HTML file and open the file with a browser, the J-DSP applet will be activated, and by pressing [Start], the J-DSP editor will load containing a **SigGen** block as shown in figure 4.

Figure 4: Our first scrip loaded

3.3. Establishing connections.

C is used to denote connections between the blocks. A <PARAM> tag containing **C** in its value will connect two blocks in the same manner as we connect two blocks by dragging from a pin of a block to a pin of another block to connect them. The possible pin numbers for the blocks are shown below.

The format of VALUE in the <PARAM> tag that establishes connection between two blocks has a general format as given below:



The connection shown above requires a code given by C-0-4-1-0. This translates to connecting the first block (number 0) pin 4 to the second block (number 1) pin 0. If we assume this is the 6th general PARAM tag (count starts from 0) the complete line is:

<param name="5" value="**C**-0-4-1-0">

We can now write a simple program to connect two blocks in J-DSP. In the code is given below observe how the <PARAM> tag in line 1 with “numCommand” as PARAM NAME, has VALUE=“3” because there are 3 general <PARAM> tags in this program.

```
0   <applet CODE="JDsp.class" width="400" height="250">
1   <param name="numCommand" value="3">
2   <param name="0" value="B0-siggen(3,1)">
3   <param name="1" value="B1-plot(5,1)">
4   <param name="2" value="C-0-4-1-0">
5   </applet>
```

The above program establishes a **SigGen** block in the (3,1) position and a **Plot** block in the (5,1) position of the editor frame. The last <PARAM> tag connects the two blocks, by connecting **SigGen** block’s 4th pin and **Plot** block’s 0th pin. Opening the html file and starting J-DSP will give the flowgram of figure 5.

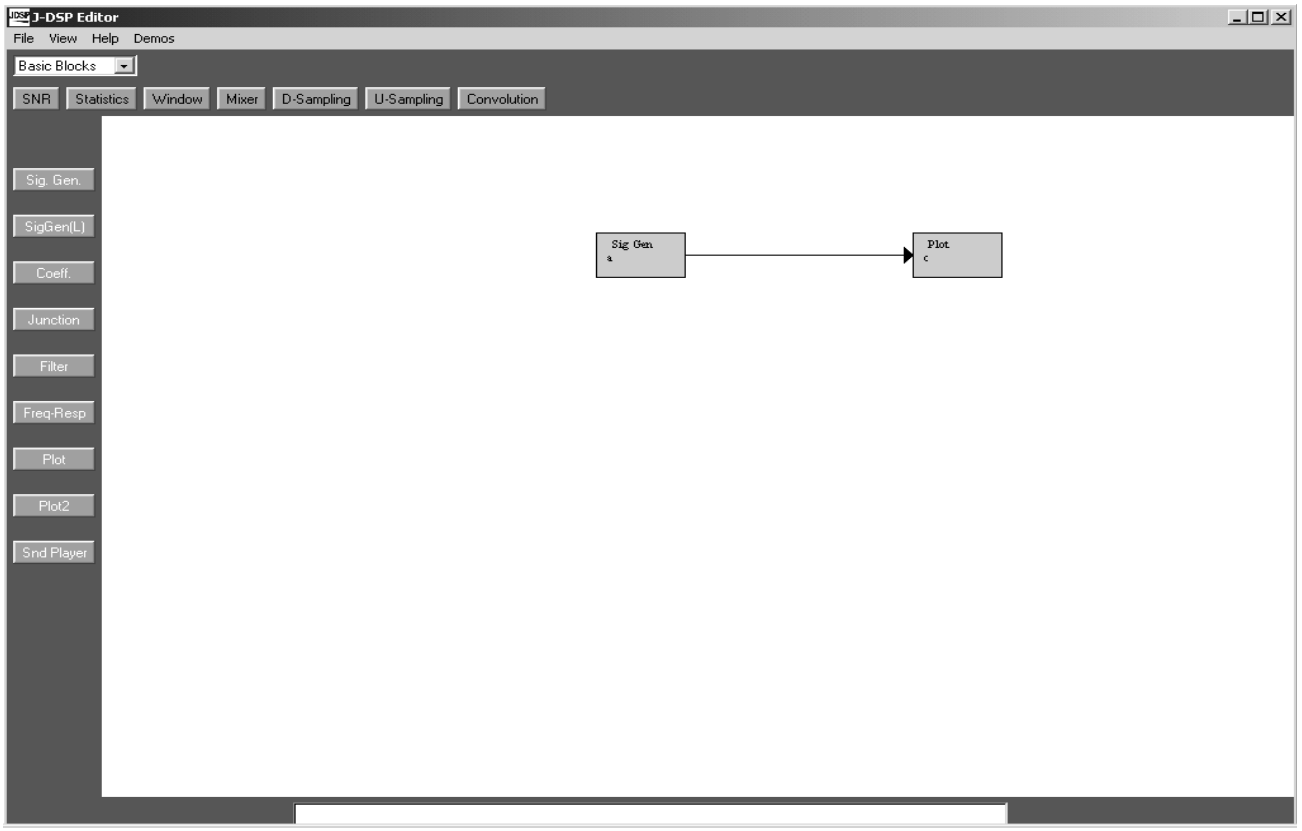


Figure 5: A flowgram with connections

3.4. Passing parameters to parts.

P is used to denote parameter passing in a specific block. A <PARAM> tag containing **P** in its value will cause the set of parameters that follow to be passed to the loaded block. The number concatenated with **P** is the number originally given to the part when created with the **B** command. For example, in the code below, line 3 shows how a signal generator part is created and given the number 0.

<param name="0" value="B0-siggen(1,3)">

Then, line 9 is used to pass parameters to the signal generator part by placing a 0 next to the P command.

Parameters to be passed to the block

<param name="2" value="P0~24,10,0,~3.0,0.9,0.0,0.2,~a,Triangular,Yes,null,~~">

Summarizing, **P** denotes that parameters are to be passed, and 0 gives the part number to which they are to be passed to. The set of parameters to be passed to the part then follows.

0	<applet CODE="JDsp.class" width="400" height="250">
1	<param name="numCommand" value="3">
2	<!-- START PARTS -->
3	<param name="0" value="B0-siggen(1,3)">
4	<!-- END PARTS -->
5	<!-- START OPEN DIALOGS -->
6	<param name="1" value="O-0">
7	<!-- END OPEN DIALOGS -->
8	<!-- part parameters -->
9	<param name="2" value="P0~24,10,0,~3.0,0.9,0.0,0.2,~a,Triangular,Yes,null,~~">
10	<!-- part parameters -->
11	</applet>

The set of parameters passed to each part differs but the form of the parameter passing line is always the same and is given by

<param name="*x*" value="P*y*~i_0,i_1,…i_n,~d_0,d_1,…d_n,~s_0,s_1,…s_n,~b_0,b_1,…b_n,~">

where *x* is the script line number and *y* is the number given to the part when it was created. Here, *i* stands for integers (e.g. 2), *d* for doubles (e.g. 3.52), *s* for strings (text) and *b* for Booleans (true or false). Note that this line has to be typed exactly as shown above with the commas (,) and tildes (~) in the exact position even when a certain type of variables is not present (notice that the two consecutive tildes at the end of line 9 are there, even though no Boolean variables are present).

For explanatory purposes table 1 describes the parameters of a signal generator (**SigGen**) block. However, since it could become cumbersome to manually pass parameters to a part, no other tables have been added and the user is urged to automatically generate scripts that pass parameters to parts. This will save both time and debugging efforts. The following paragraphs describe how to use table 1.

As mentioned earlier, the line necessary to pass parameters to a part has the following form:

<param name="*x*" value="P*y*~i_0,i_1,…i_n,~d_0,d_1,…d_n,~s_0,s_1,…s_n,~b_0,b_1,…b_n,~">

All the information that is to replace the i, d, s and b is given in table 1 specifically for the signal generator block. The table is divided into four rows, for the variable name, type, position in the code and the allowed range.

The variable name is a short description of the variable, and is not used in the code. The type describes whether the variable is an integer, a double, a string or a Boolean. The position is the exact location where the variable should be placed in the code. It is given by a combination of a letter and a number subscripted after it. The letter can be i, d, s or b for integer, double, string and Boolean respectively. The integer denotes the position, starting with 0. The range is the allowed assortment of values the variable is allowed to take. All variable names with a * and a number in parenthesis depend on a particular selection of signal type but nonetheless should be used.

By replacing the necessary variables in the above generic script line you can get the necessary code to pass parameters to a part.

Variable	Pulse Width	Period	Time shift	Gain (amplitude)	Exponential base (*1)
Type	Integer	Integer	Integer	Double	Double
Position	i_0	i_1	i_2	d_0	d_1
Range	1-256	≥ 0	≥ 0	≥ 0	>0

Variable	Mean(*2)	Frequency(*3)	Part name	Signal type	periodic
Type	Double	Double	String	String	String
Position	d_2	d_3	s_0	s_1	s_2
Range	≥ 0.0	≥ 0.0	A 5 digit alphanumeric word	Rectangular Triangular Delta Exponential(*1) Sinusoid(*3) Sinc Random(*2) Self Defined	Yes No

Variable	Distribution(*2)				
Type	String				
Position	s_3				
Range	Null Uniform Gaussian Rayleigh				

Table 1: Signal Generator parameters table

The parameter passing line can be skipped if no parameters are to be passed to the part, which then loads with its default parameters.

Finally, figure 6 shows a J-DSP script for creating a simulation using a filter and a coefficient block.

```
<applet CODE="JDsp.class" width="400" height="250">

<param name="numCommand" value="12">
```

```
<!-- START PARTS -->
<param name="0" value="B0-siggen(1,3)">
<param name="1" value="B1-filter(3,3)">
<param name="2" value="B2-coeff(3,5)">
<param name="3" value="B3-plot(5,3)">
<!-- END PARTS -->

<!-- START CONNECTIONS -->
<param name="4" value="C-2-3-1-2">
<param name="5" value="C-0-4-1-0">
<param name="6" value="C-1-4-3-0">
<!-- END CONNECTIONS -->

<!-- START OPEN DIALOGS -->
<param name="7" value="O-3">
<!-- END OPEN DIALOGS -->

<!-- part parameters -->
<param name="8" value="P0~20,10,0,~1.0,0.9,0.0,0.2,~a,Rectangular,No,null,~~">
<param name="9" value="P1">
<param name="10" value="P2~~1.0,1.8,-
2.0,3.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,~c,~~">
<param name="11" value="P3~~~d,cont.,Magn.,linear,~false,~">
<!-- part parameters -->

</applet>
```

Figure 6

3.5. Opening Dialog Windows.

O is used to open the dialog box of a block. The effect is the same as double clicking on a block in the J-DSP editor frame. The number of the block has to be specified here. This is the number that we assigned to the block when it was established. For example, **O**-1 opens the dialog box of the second block of the flowgram, because '1' is the number of the second block. The complete line is:

<param name="9" value="**O**-1">

Now we are ready to write a simple program to connect two blocks in J-DSP and open their dialog boxes to observe input and output plots. Here is the necessary code:

0	<applet CODE="JDsp.class" width="400" height="250">
1	<param name="numCommand" value="5">
2	<param name="0" value="**B**0-siggen(3,1)">
3	<param name="1" value="**B**1-plot(5,1)">
4	<param name="2" value="**C**-0-4-1-0">
5	<param name="3" value="**O**-0">
6	<param name="4" value="**O**-1">
7	</applet>

The above program establishes a **Sig Gen** block in the (3,1) position and a **Plot** block in the (5,1) position of the editor frame. The <PARAM> tag in line 4 connects the two blocks, by connecting **Sig Gen** block's 4th pin and **Plot** block's 0th pin. The last 2 <PARAM> tags in lines 5 and 6 open the dialog boxes of the **SigGen** block (block number '0') and the **Plot** block (block number '1') respectively. Observe that the <PARAM> tag with "numCommand" as PARAM name, has VALUE= "5" because there are 5 general <PARAM> tags in this program.

Opening the HTML file and starting J-DSP will give the following flowgram, as show in figure 7.

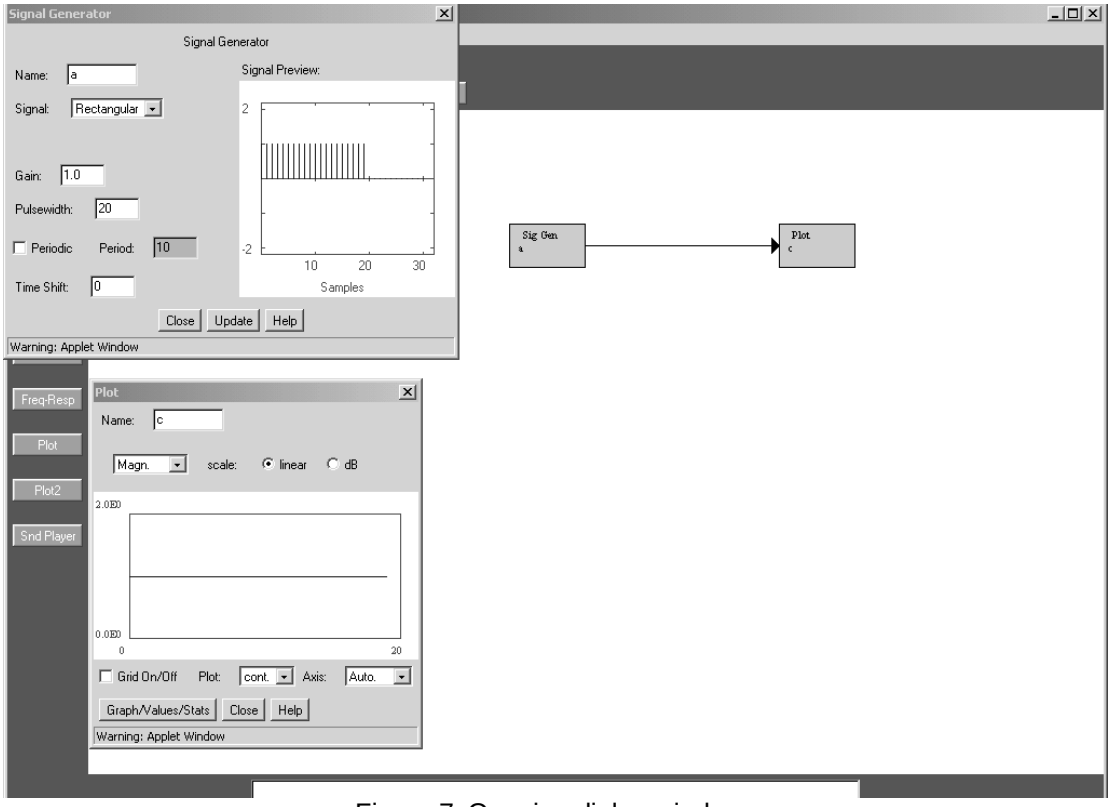


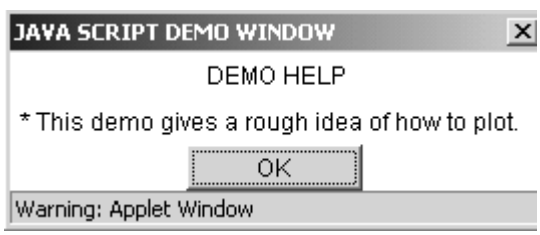
Figure 7: Opening dialog windows

3.6. Help Dialogs.

The asterisk sign (*) followed by any message generates a help dialog box containing the message. An example is given below:

<param name="15" value="***** 1. This demo gives a rough idea of how to plot ">

This line establishes the following dialog box in the J-DSP editor:



3.7. An example

The following example includes several of the J-DSP scripts parameters. The resulting J-DSP editor window is shown in figure 8.

```
<applet CODE="JDsp.class" width="400" height="250">

<param name="numCommand" value="14">

<!-- START PARTS -->
<param name="0" value="B0-siggen(1,2)">
<param name="1" value="B1-filter(2,2)">
<param name="2" value="B2-plot(4,2)">
<param name="3" value="B3-coeff(2,5)">
<!-- END PARTS -->

<!-- START CONNECTIONS -->
<param name="4" value="C-0-4-1-0">
<param name="5" value="C-1-4-2-0">
<param name="6" value="C-3-3-1-2">
<!-- END CONNECTIONS -->

<!-- START OPEN DIALOGS -->
<param name="7" value="O-0">
<param name="8" value="O-2">
<param name="9" value="O-3">
<!-- END OPEN DIALOGS -->

<!-- START PART PARAMATERS. * DO NOT MODIFY! * -->
<param name="10" value="P0~30,10,0,~1.0,0.9,0.0,0.2,~a,Triangular,No,null,~~">
<param name="11" value="P1">
<param name="12" value="P2~~~c,linear,Magn.,cont.,~false,~">
<param name="13" value="P3~~1.0,1.6,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,2.0,-
2.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,~d,~~">
<!-- END PART PARAMATERS -->

</applet>
```

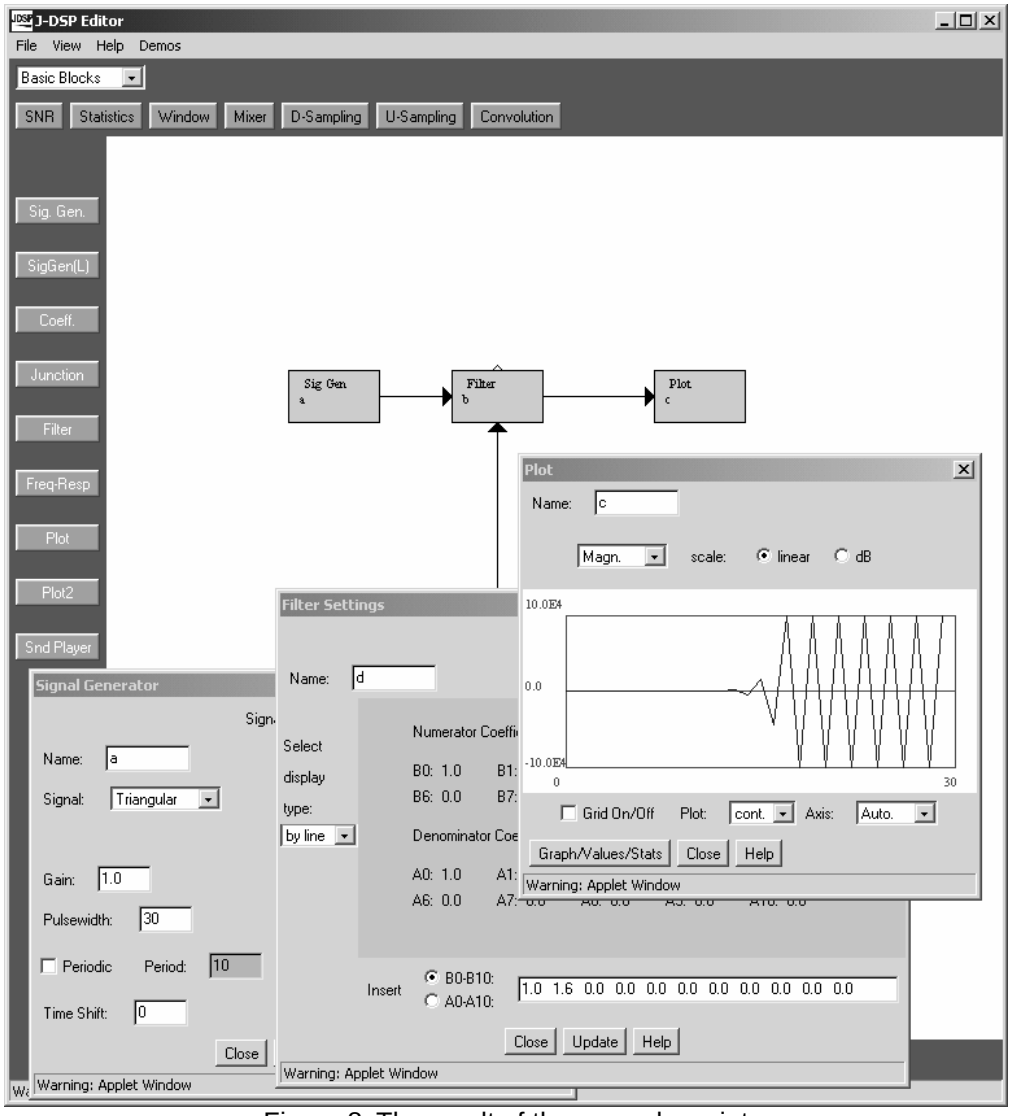


Figure 8: The result of the example script