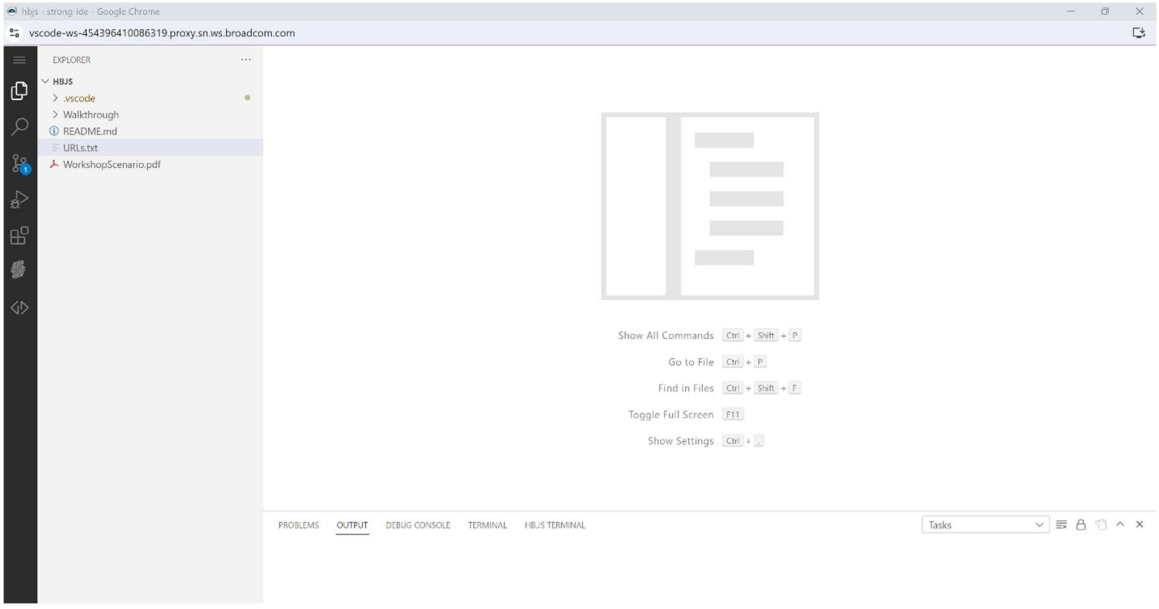
Hostbridge HBJS Workshop IV

In this workshop you will create a web service that orchestrates multiple CICS transactions, creating a business-process oriented API.

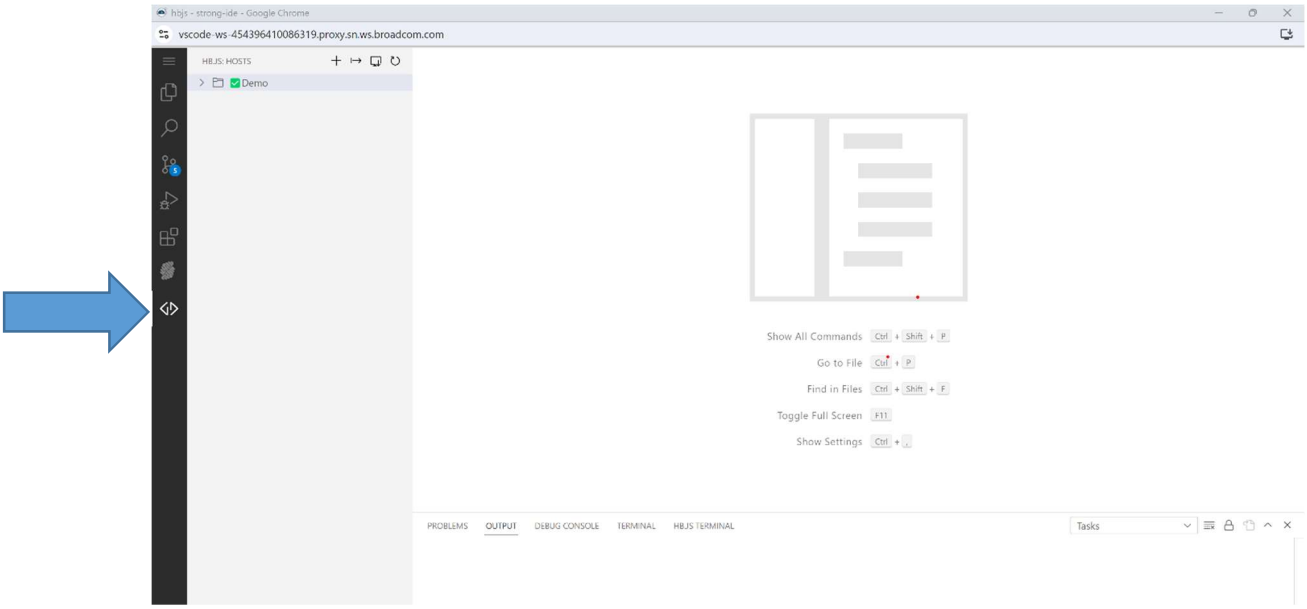
There will be instructions provided to help you navigate to the following screen. Let us know if you have any questions.



Configure:

The HostBridge extension has already been installed for you. The next steps will walk you through configuring it for this workshop.

Click on the HB.js icon shown below to navigate to the following screen.

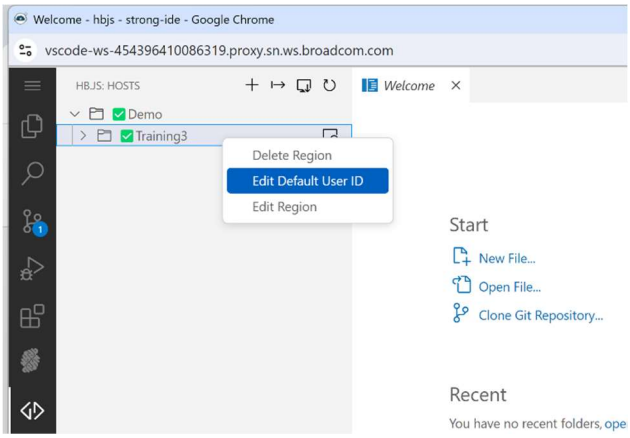


You should be able to see the Demo host already configured for you.

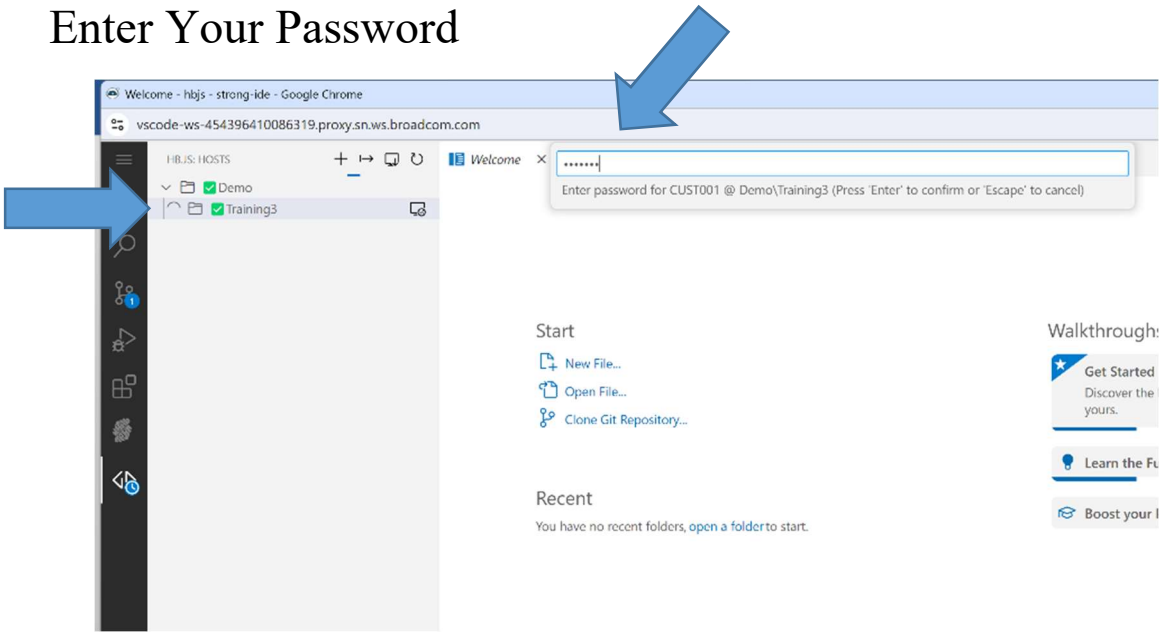
Click on the Host to display the region, and then right click on the region to set your default user ID “Edit Default User ID”. Then click on the region to enter your Password.
Your credentials are:

UserName & Password : CUST###

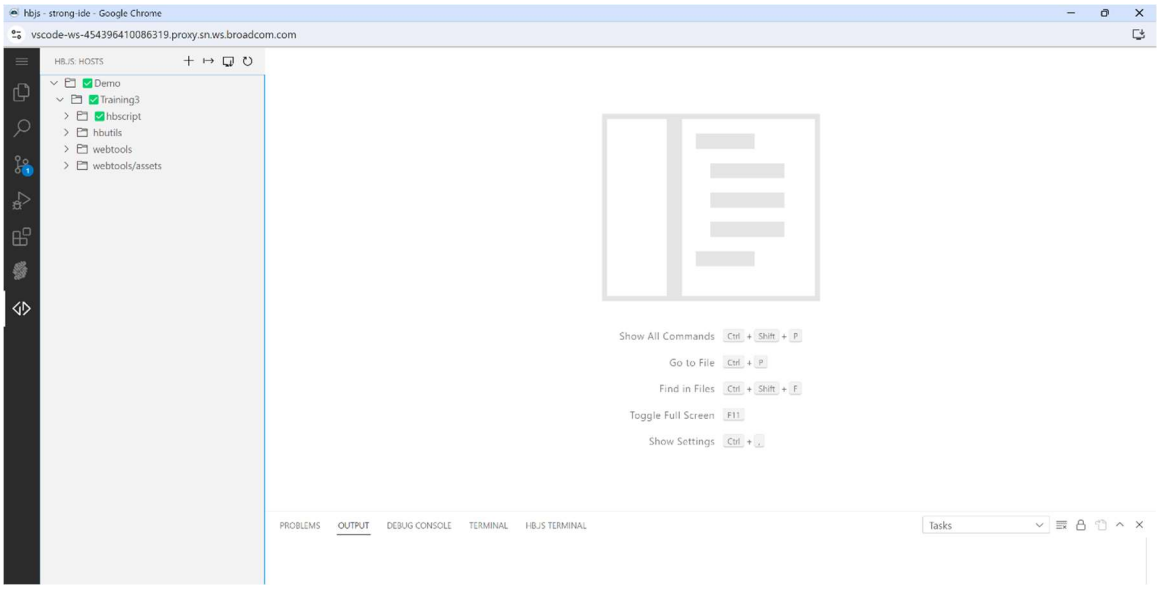
Edit Default User ID



Enter Your Password



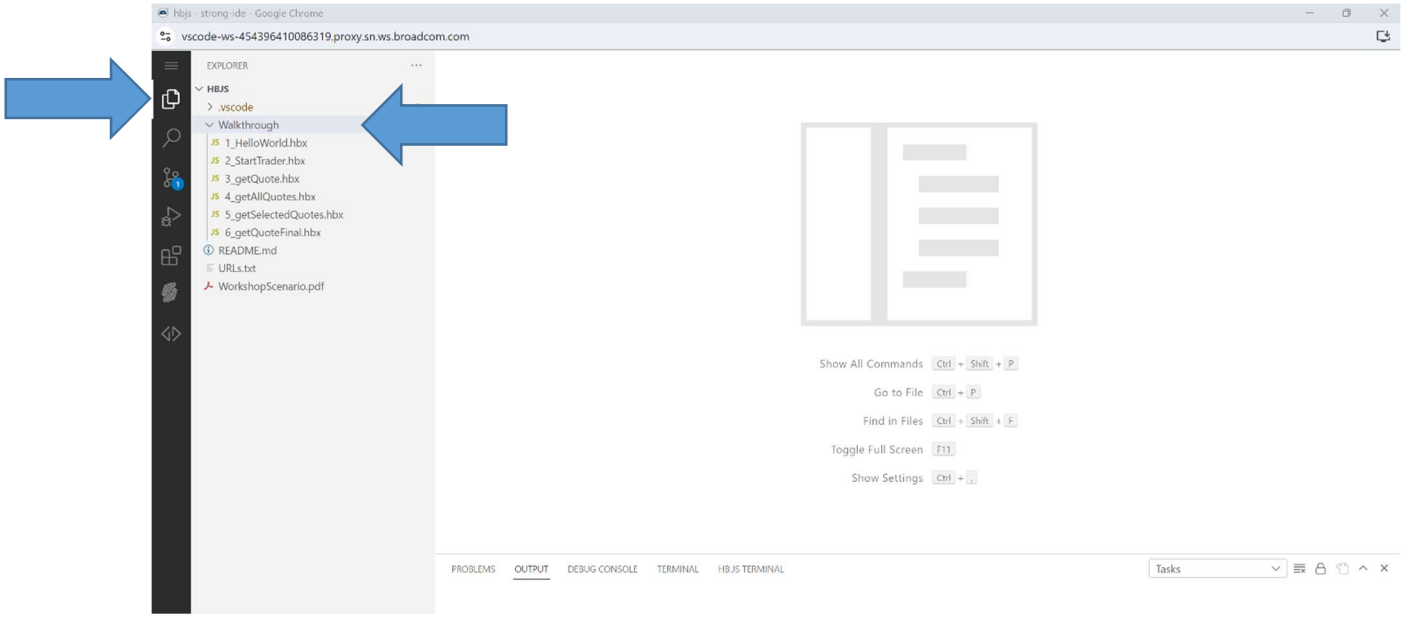
Now you should see several repositories as shown below:



If you do not see the green check marks shown above, right click on the “hbscript” repository and select “Set Default Repository”. You should now see the green check mark on your Host > Region > and Repository.

Get Exercises:

You are now ready to access and begin the workshop exercises. Navigate to the explorer tab and expand the Walkthrough folder. The following screen is returned.



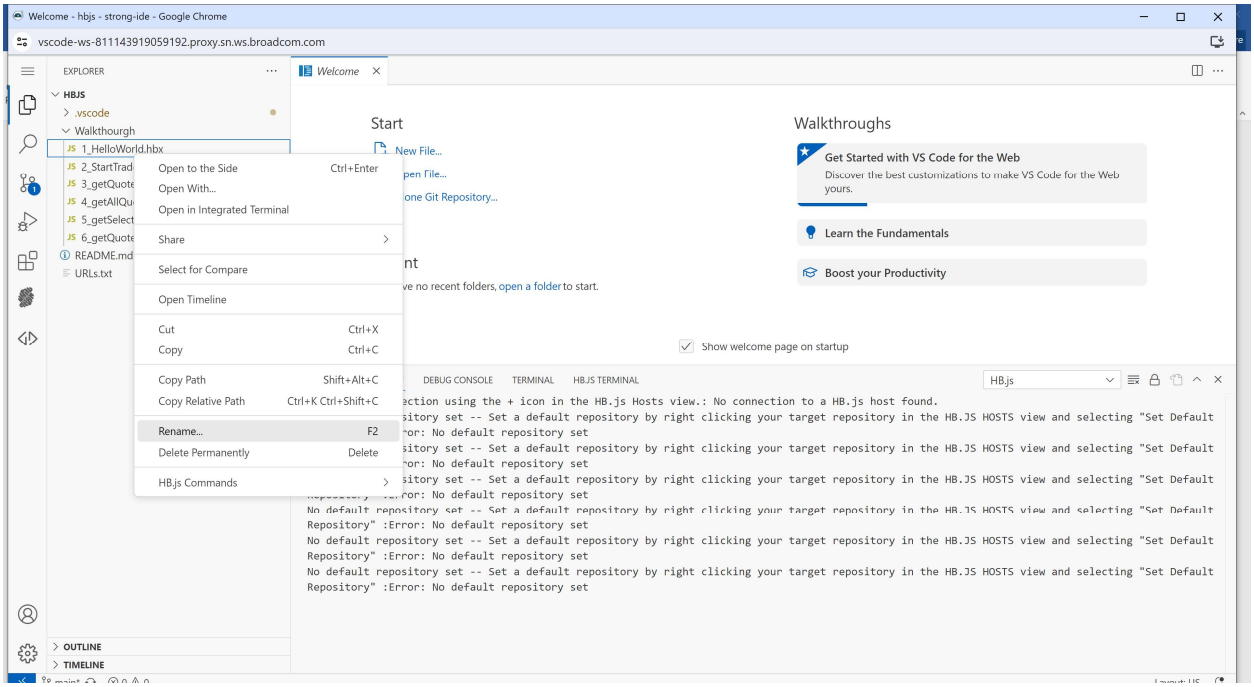
Workshop #1

The first workshop exercise is to change a line of Java Script. In the folder Walkthrough you will notice two files.

1-HelloWorld.hbx

2-StartTrader.hbx

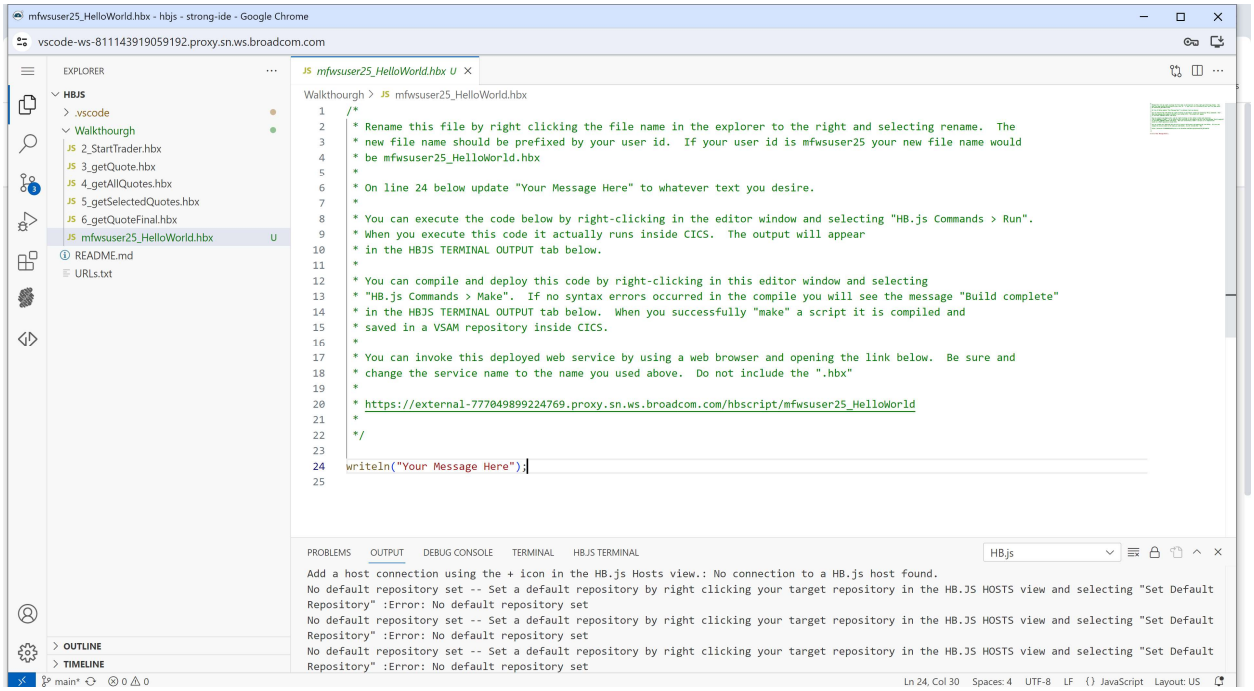
Rename the 1-HelloWorld.hbx, by right clicking the file and then selecting rename.



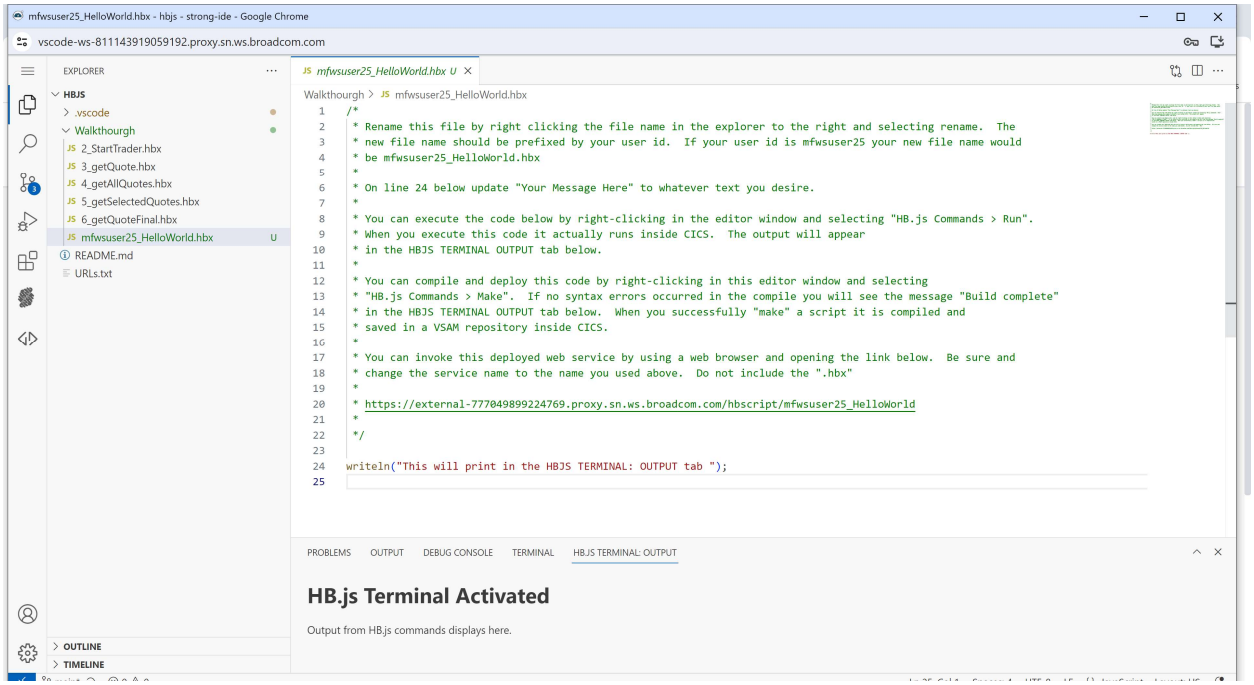
The new file name should be prefixed by your assigned userid mfwsuser## where the ## is your assigned number. So the file name should look like:

mfwsuser##_HelloWorld.hbx

The next step is to click on the file to open it and modify line 25:



Change “Your Message Here” to the message you desire.



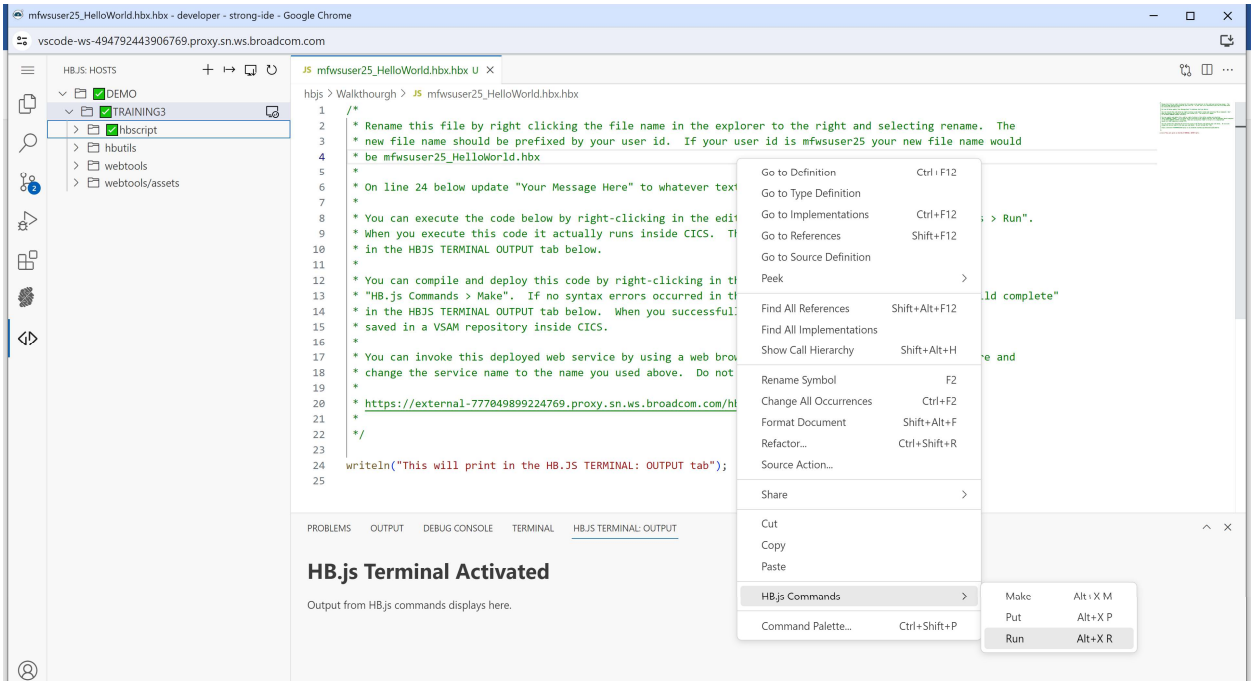
To execute the script right click anywhere inside the code and hover over HB.js Commands, notice three options:

Make – Places the script on the mainframe and compiles it. It will only put the script on the mainframe if there is a clean compile.

Put – Places the script on the mainframe

Run – executes the script

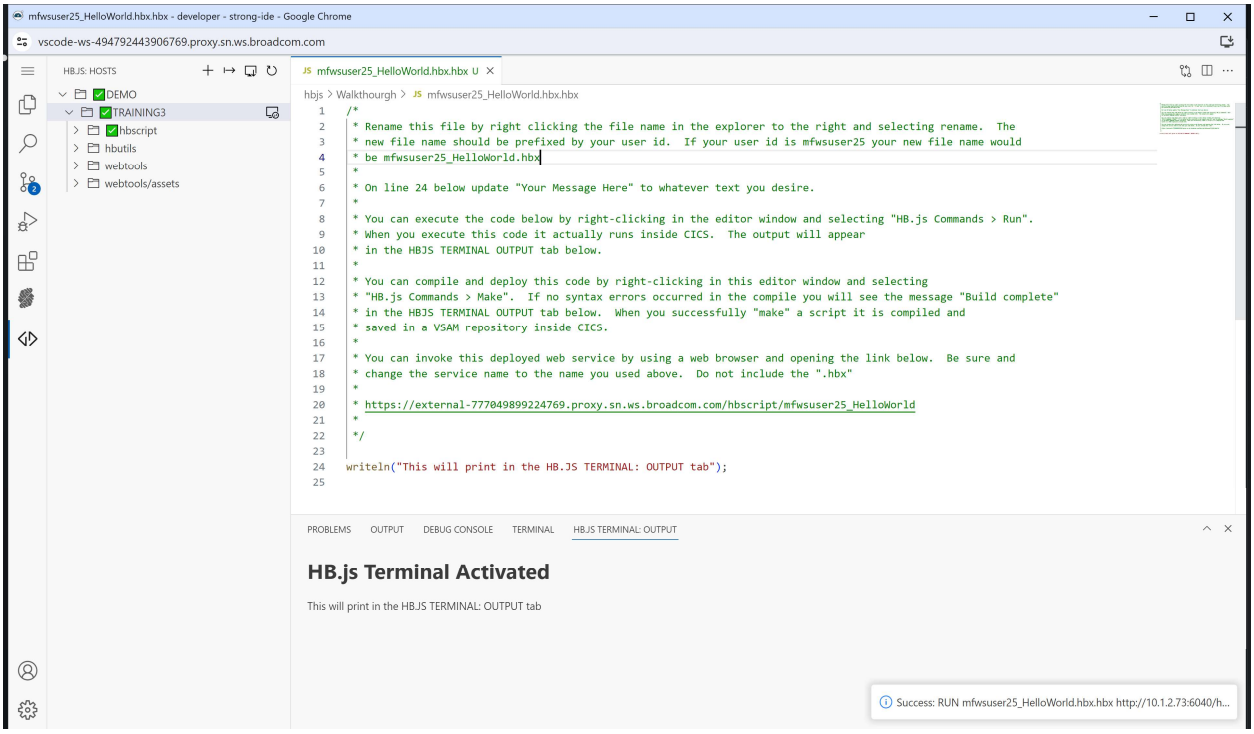
You can also use the listed hot keys if that is your preference.



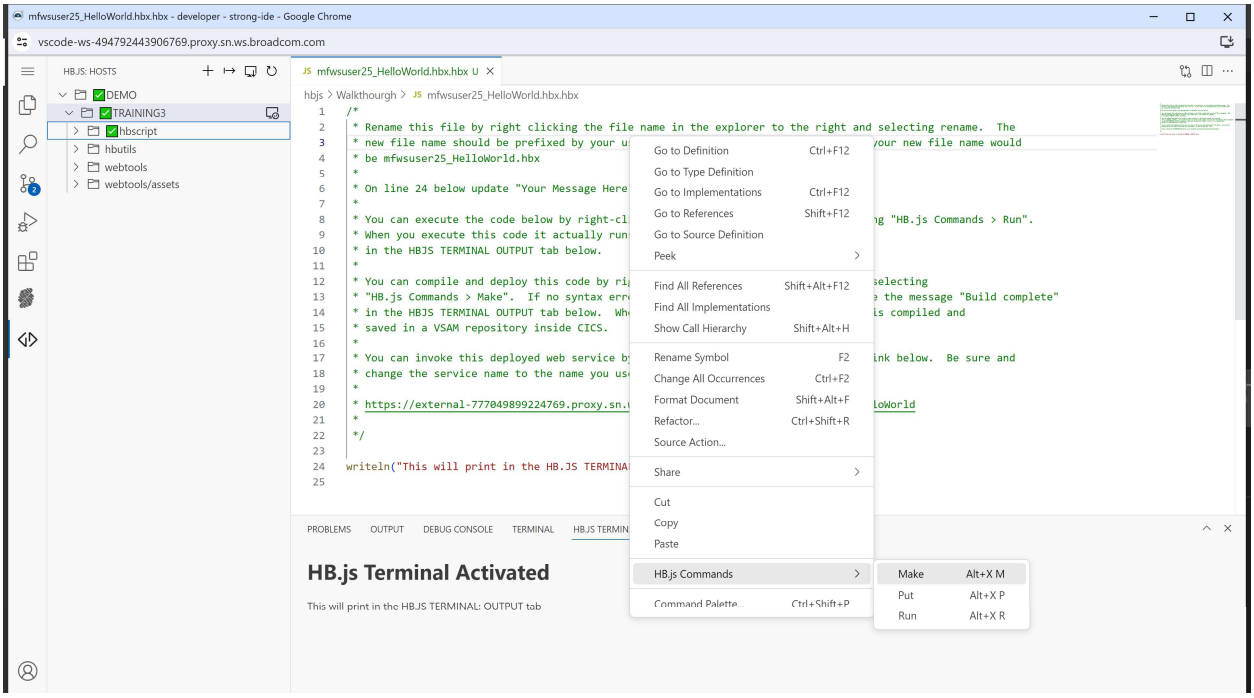
Select the HB.JS TERMINAL:OUTPUT tab and then Click HB.js Commands > Run (the tab need only be selected once per session, after that it has been activated and will open for you when output is printed to it).

When you execute the script it is actually running inside the CICS region.

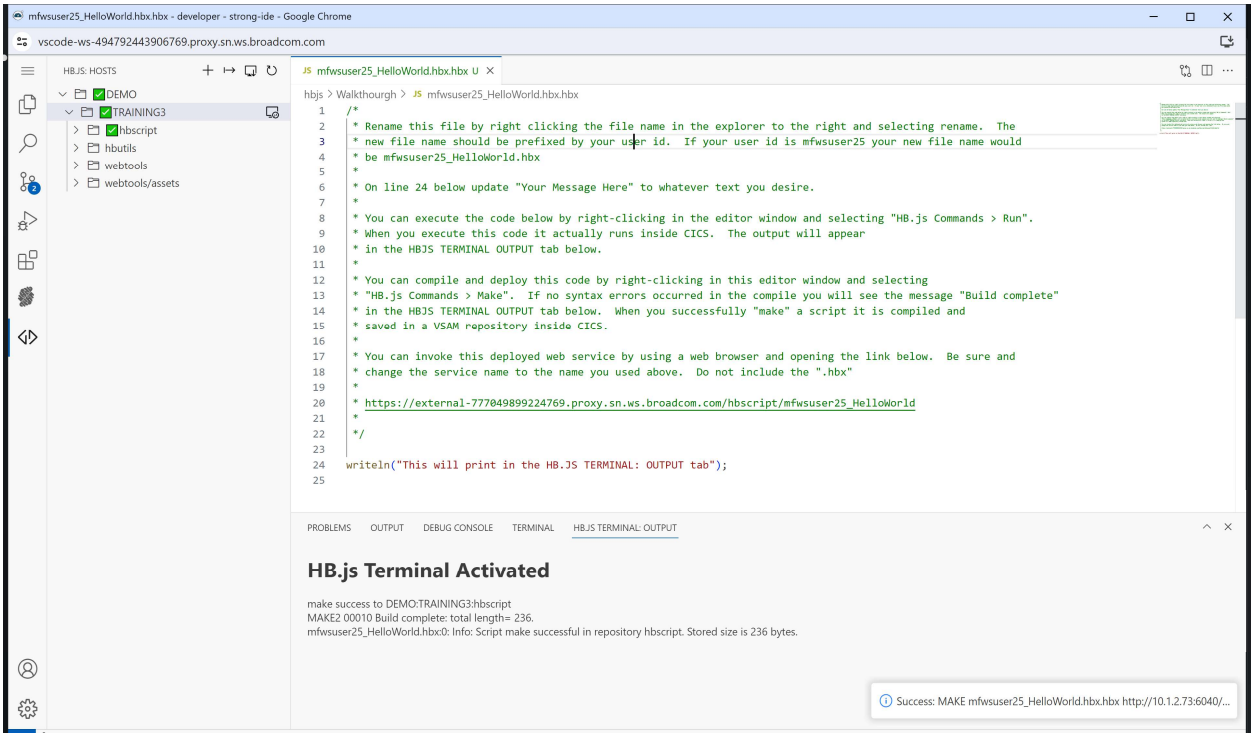
See your message in the HB.JS TERMINAL:OUTPUT and the Success prompt on the bottom right.



Next right click inside the code and click Make.



If there are no errors, you will have compiled and copied the code to the mainframe. Notice the message in the HB.JS TERMINAL:OUTPUT.



Next would be to invoke this service using a Chrome web browser outside of the strong network and opening the following link.

Remember to change the ## to your user number for today's workshop.

If you cannot copy this link, it also exists in your HelloWorld script.

https://external-9430252906936663.proxy.sn.ws.broadcom.com/hbscript/mfwsuser##_HelloWorld

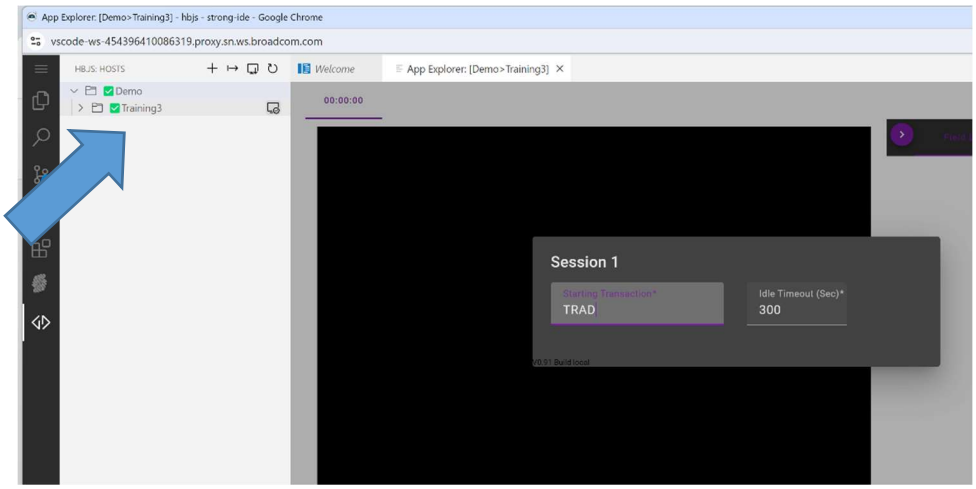
If you see your chosen input printed to the screen, congratulations, you have completed the first exercise.

Workshop #2

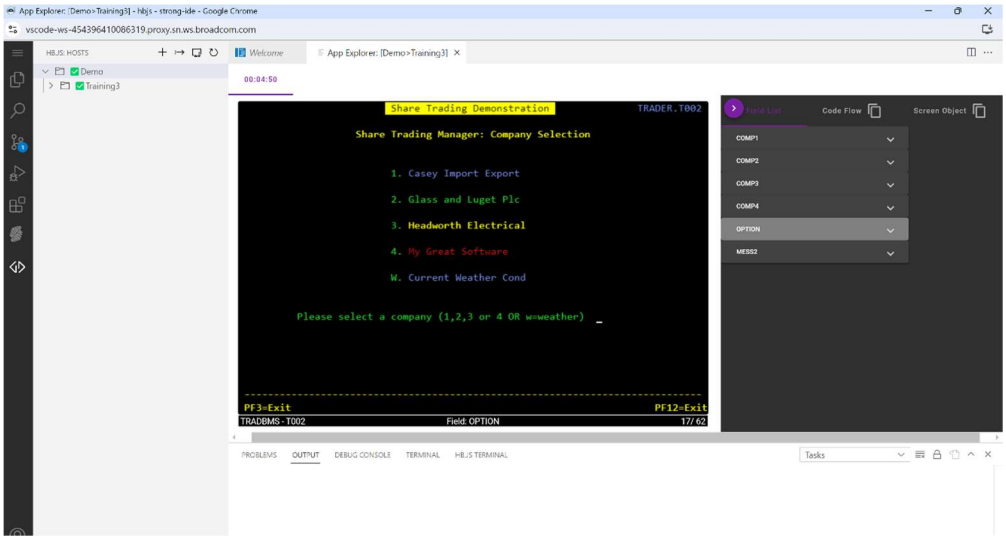
Repeat the renaming steps in Workshop #1 on 2-StartTrader.hbx.

Now click on the icon to the right of your Training 3 Region to access the Application Explorer.

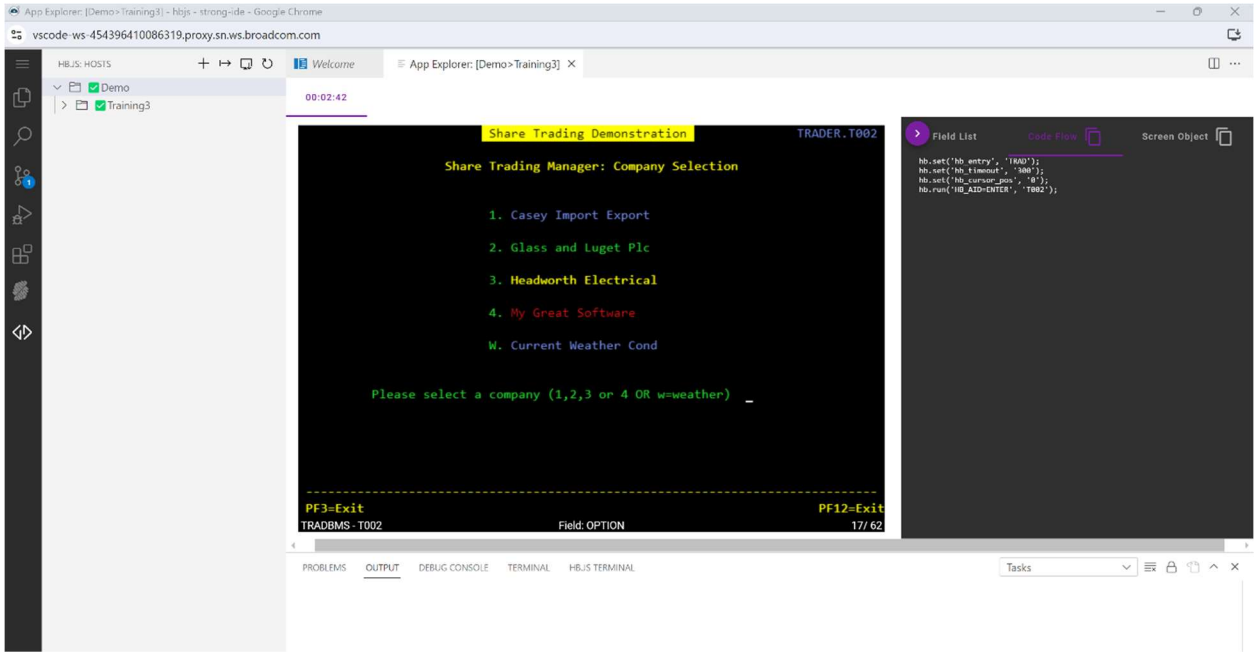
If you are prompted for your username & password, they are: CUST##



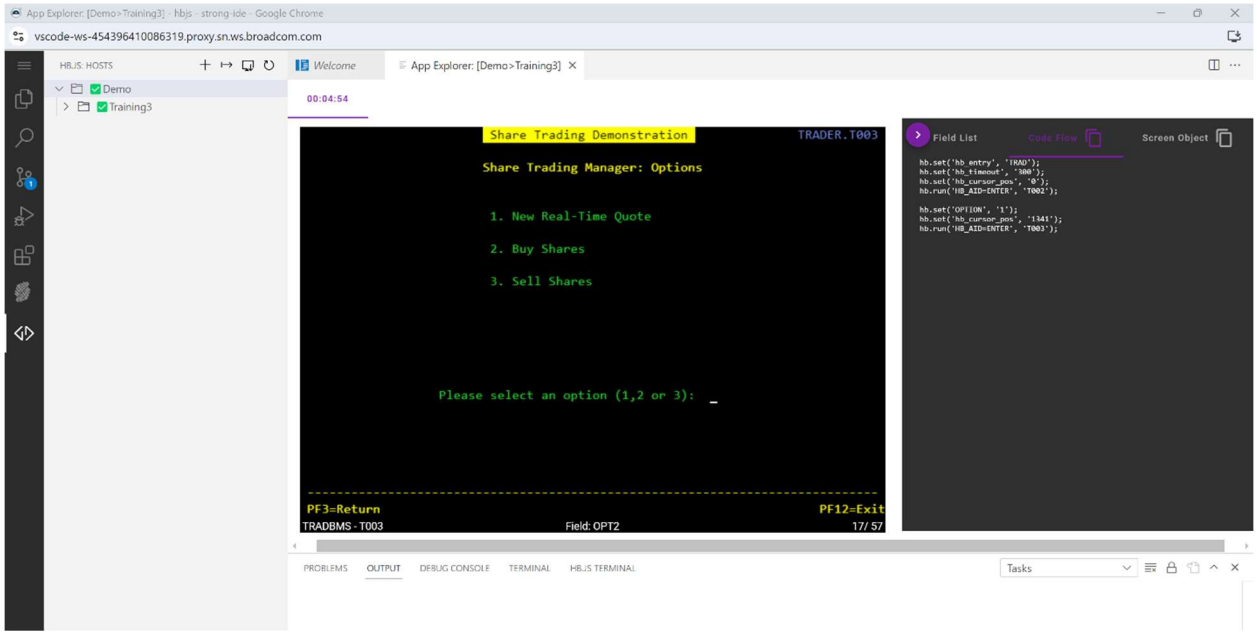
In the Starting Transaction field type TRAD and press enter to navigate to the below screen.



One feature HB.js has is its Code Flow capabilities. Click on the tab to access the code equivalent of the steps we will navigate.

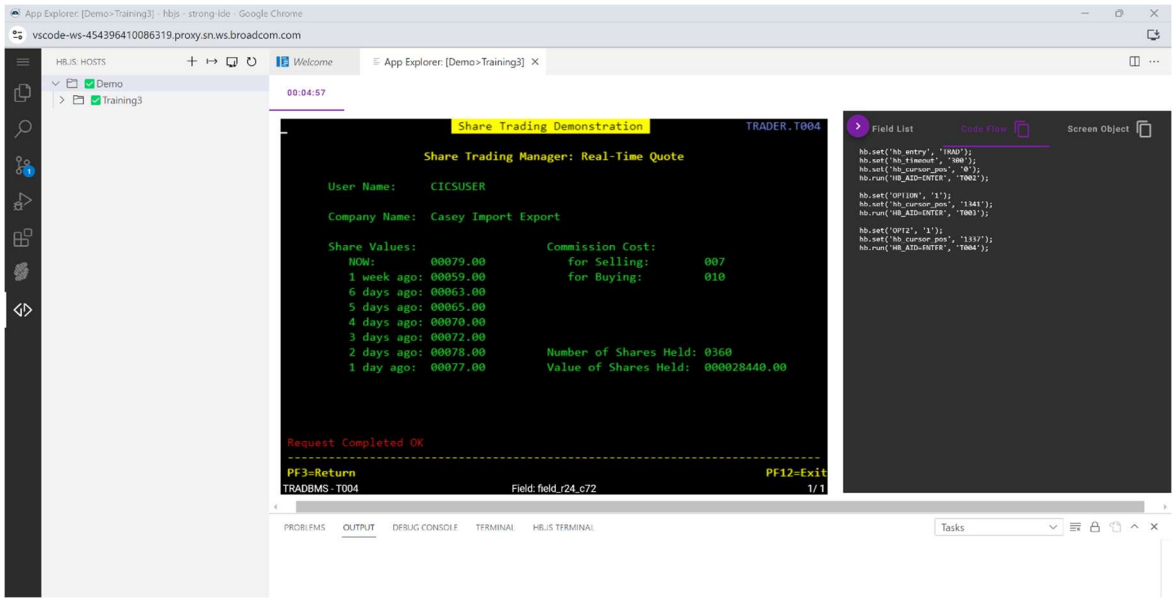


At this point you are walking through an application and recording the steps. Enter a 1 on the Please select a company field and press enter.

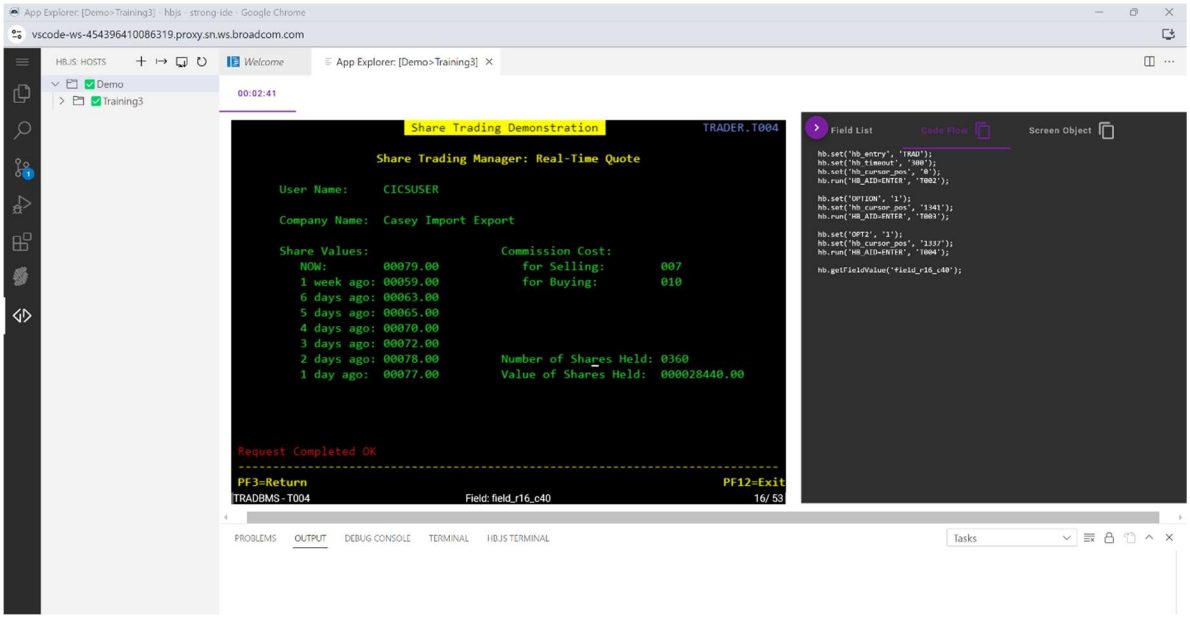


The screen changes to Share Trading Demonstration but in the Code Flow you see the HB.js commands.

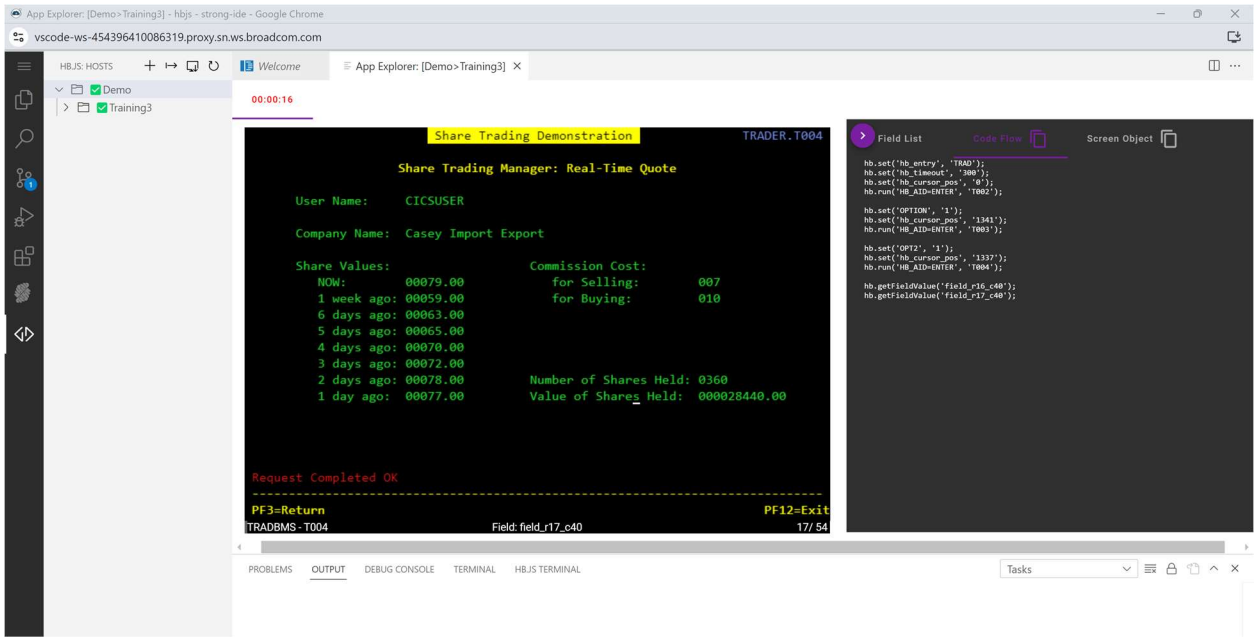
Enter a 1 on the Please select an option field and press enter. See how the Code Flow has changed.



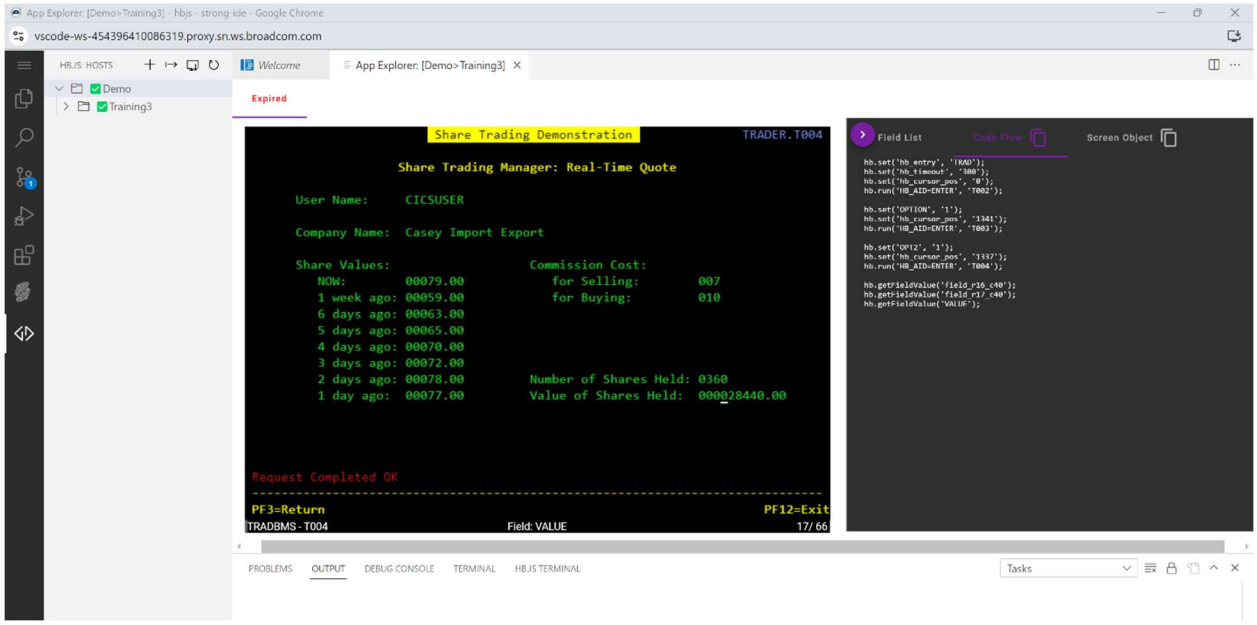
Place your courser on the Number of Shares Held field and double click. This will add the field to the Code Flow.



Repeat again with the Value of Shares Held field.



Now place your courser on the value of the Value of Shares Held field which is 000028440 and double click to add that value to the code flow.



At this point HB.js has written the code for you. All you have to do is copy it and paste it into the script between the following lines:

// ---------- Paste the copied code below here

// ---------- Paste the copied code above here

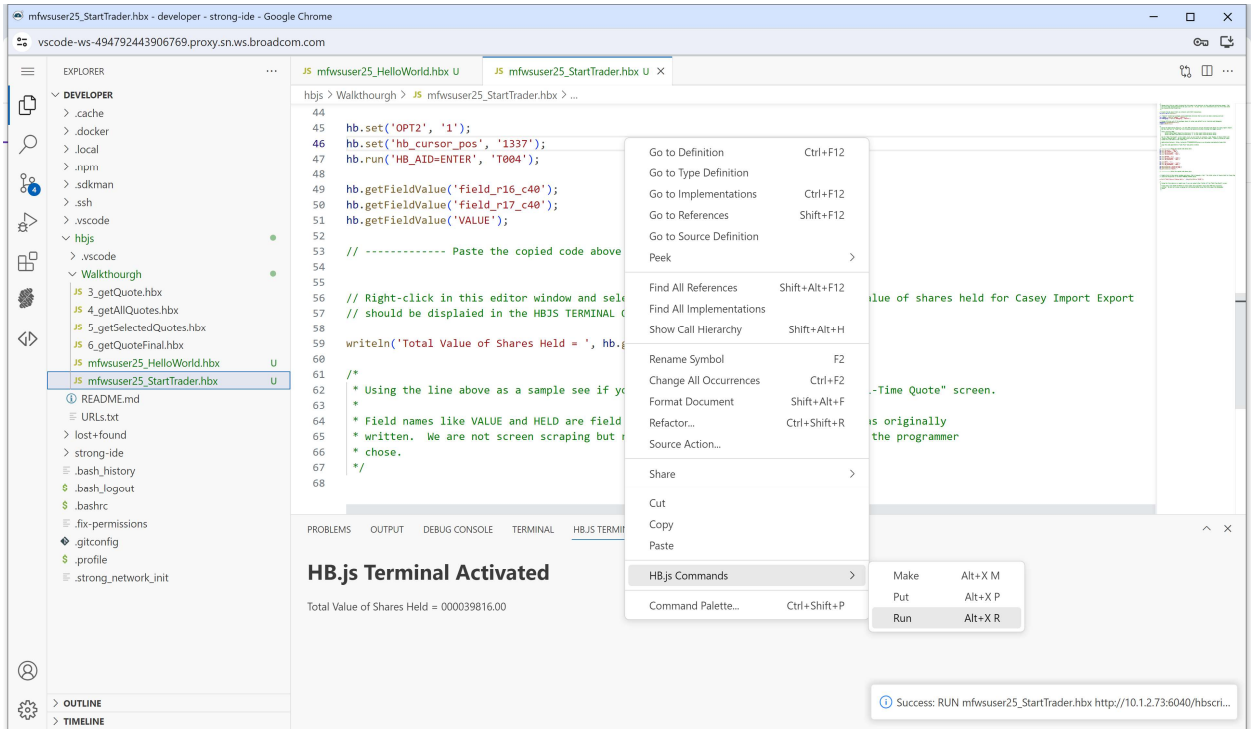
So it will look like the following:

// ---------- Paste the copied code below here

```
hb.set('hb_entry', 'trad');
hb.set('hb_timeout', '300');
hb.set('hb_cursor_pos', '0');
hb.run('HB_AID=ENTER', 'T002');

hb.set('OPTION', '1');
hb.set('hb_cursor_pos', '1341');
hb.run('HB_AID=ENTER', 'T003');

hb.set('OPT2', '1');
hb.set('hb_cursor_pos', '1337');
hb.run('HB_AID=ENTER', 'T004');

hb.getFieldValue('field_r16_c40');
hb.getFieldValue('field_r17_c40');
hb.getFieldValue('VALUE');
```

// ---------- Paste the copied code above here

At this point you can right click the script and issue the HB.js Commands > Run



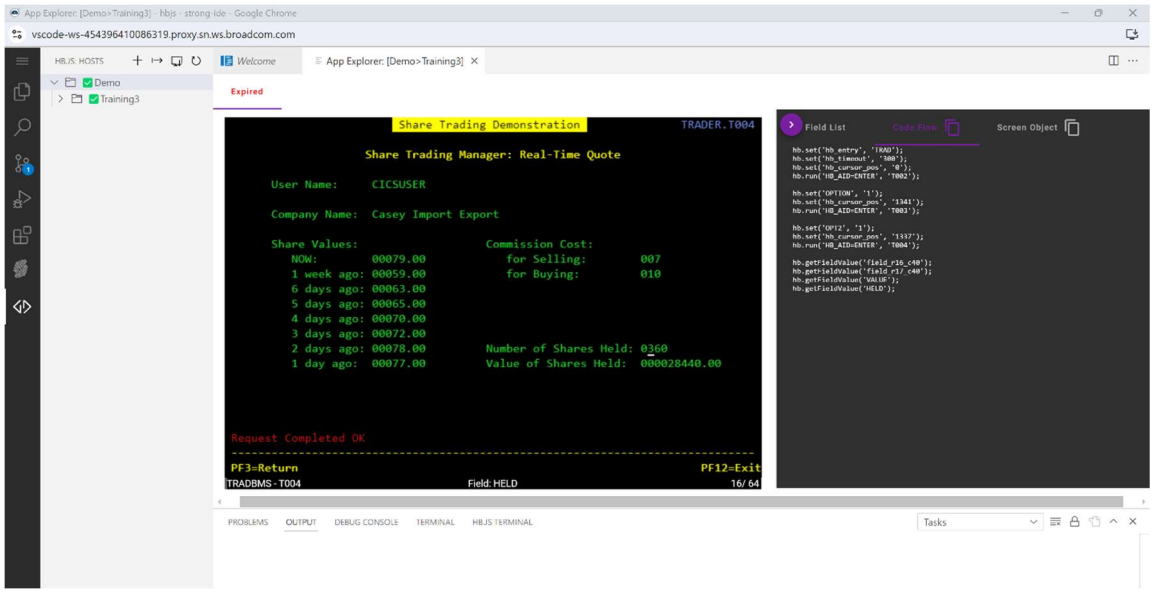
You should see Total Value of Shares Held = 000028440.00

Once you have completed this part of the exercise add another output field Number of Shares Held. To do this you

have to go back to the Application Explorer screen and double click the value for Number of Shares Held field.

When you double click the value you will see the HB.js command added to the Code Flow. Paste it directly below line:

hb.getFieldValue('VALUE');



hb.getFieldValue('HELD');

In the script let's add the HELD value to the output. To do this repeat the following line, changing the VALUE to HELD:

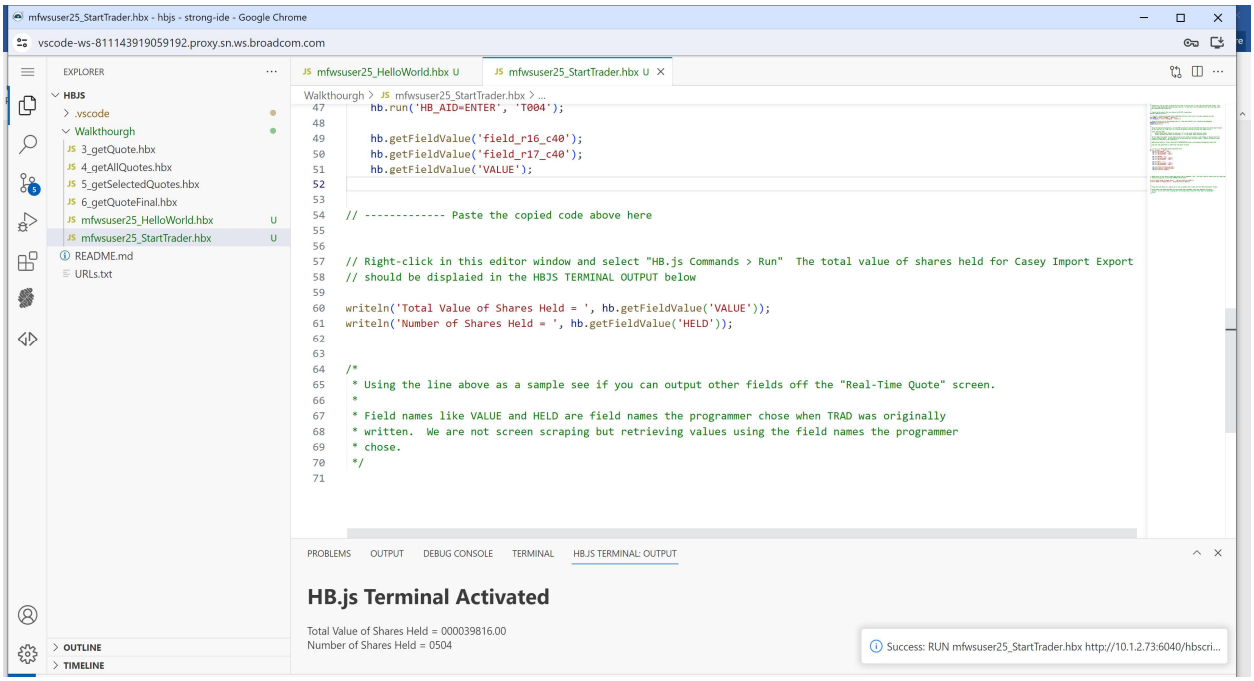
writeln('Total Value of Shares Held = ', hb.getFieldValue('VALUE'));

When you are finished it should look like the following.

writeln('Total Value of Shares Held = ', hb.getFieldValue('VALUE'));

writeln('Number of Shares Held = ', hb.getFieldValue('HELD'));

Repeat the above instructions on running the script and notice second line of output.



The complete script should look close to the following:

```
/*
 * Rename this file by right clicking the file name in the explorer to the right and selecting rename.  The
 * new file name should be prefixed by your user id.  If your user id is mfwsuser## your new file name would
 * be mfwsuser##_StartTrader.hbx
 */

// Create the hb object that can interact with CICS transactions
let hb = new HB.Session();

// "require" JavaScript packages with predefined utilities that we will use when creating services
let common = require('common', 'hbutils');
let debugging = require('debugging', 'hbutils');
```

```
// Using utilities out of the packages above to setup some default error checking
and debugging
debugging.checkDebugControl();
common.hbRunProto();

/*
 * Using the Application Explorer, run the TRAD transaction and get the Real-Time
Quote for Casey Import Export.
 * Be sure and turn on "Code Flow" by clicking the purple arrow and clicking the
toggle switch.
 *
 * After starting TRAD:
 *      Select Casey Import Export by entering a "1" in the input field and press
enter.
 *      Select New Real-Time Quote by entering a "1" in the input field and press
enter.
 *
 * On the "Real-Time Quote" screen double click on any fields of interest, like
"Number of Shares Held" and
 * "Value of Shares Held".  Be sure and click on the values not the field
labels.  Double clicking on these
 * fields will add them to the "Code Flow".
 *
 * Copy the code generated in "Code Flow" and paste it below
 */

// ------------- Paste the copied code below here
hb.set('hb_entry', 'trad');
    hb.set('hb_timeout', '300');
    hb.set('hb_cursor_pos', '0');
    hb.run('HB_AID=ENTER', 'T002');

    hb.set('OPTION', '1');
    hb.set('hb_cursor_pos', '1341');
    hb.run('HB_AID=ENTER', 'T003');

    hb.set('OPT2', '1');
    hb.set('hb_cursor_pos', '1337');
    hb.run('HB_AID=ENTER', 'T004');

    hb.getFieldValue('field_r16_c40');
    hb.getFieldValue('field_r17_c40');
    hb.getFieldValue('VALUE');
```

```
// ------------- Paste the copied code above here


// Right-click in this editor window and select "HB.js Commands > Run"  The total
value of shares held for Casey Import Export
// should be displaied in the HBJS TERMINAL OUTPUT below

writeln('Total Value of Shares Held = ', hb.getFieldValue('VALUE'));
writeln('Number of Shares Held = ', hb.getFieldValue('HELD'));


/*
 * Using the line above as a sample see if you can output other fields off the
"Real-Time Quote" screen.
 *
 * Field names like VALUE and HELD are field names the programmer chose when TRAD
was originally
 * written.  We are not screen scraping but retrieving values using the field
names the programmer
 * chose.
 */
```

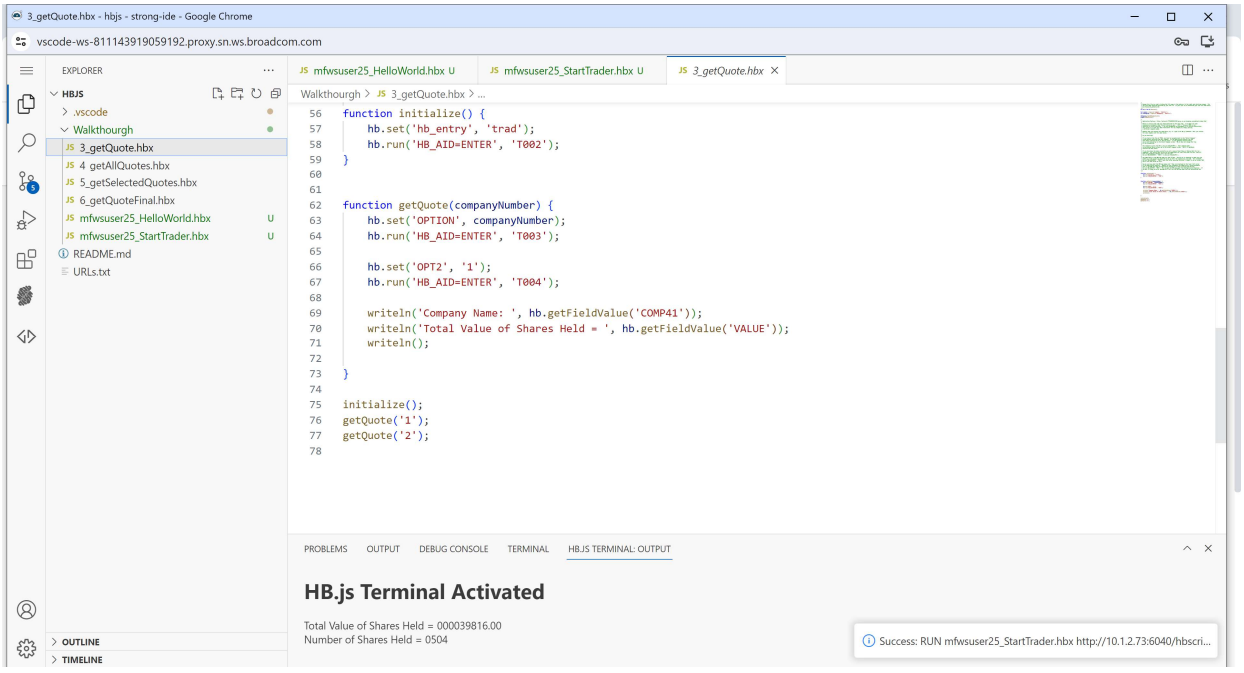
Congratulations you have completed Workshop #2

Workshop #3

In this workshop you will have to change a script to execute the TRAD transactions twice for two companies. To do this a new HB.js utility function is added which will allow you to issue a PF3 command to back up one screen at a time.

Repeat the renaming steps in Workshop #1 on 3-getQuote.hbx.

As you review the script you will notice that starting at line 62 there is a function called getQuote and it takes companyNumber in as input. But as you walk through the code it will fail because of the missing F3 or PF3 command. Also notice how a screen name check has been added ‘T003’ and ‘T004’. So when a value is entered on screen to change screens, this check will make sure you are on the correct screen.



The complete script should look close to the following:

```
/*
 * Rename this file by right clicking the file name in the explorer to the right
and selecting rename.  The
 * new file name should be prefixed by your user id.  If your user id is
mfwsuser## your new file name would
 * be mfwsuser##_getQuote.hbx
 */
let hb = new HB.Session();

let common = require('common', 'hbutils');
let debugging = require('debugging', 'hbutils');

debugging.checkDebugControl();
common.hbRunProto();

/*
 * Below is close to the code you should have had in the last step.  I've taken
out some
 * unnecessary statements like "hb.set('hb_cursor_pos', '1337');" because TRAD is
not
 * sensitive to cursor position.  I've also wrappered the statements in a
function definitions.
 * This allows us to get the "Real-Time Quote" for any company by simply passing
 * a different company number.
 *
 * However when you execute the code below (try it: right click HB.js Commands >
Run) you receive
 * the same company with the same values.
 *
 * Do you know why?
 *
 * If you look at the flow of TRAD, you have to navigate back to the "Select
Company"
 * screen before selecting the next company.  On the Application Explorer you see
 * that means pressing the F3 or PF3 key twice.  In the getQuote function below
 * we are not backing up to the Select Company screen.  HB.js does not change the
flow
 * of the transaction.
 *
 * The command to press the PF3 is hb.run('HB_AID=PF3');  That command needs
 * to be entered twice to back up to the select company screen.  Add it to
getQuote
 * function now and retest.
```

```
 *
 * If you have done the above correctly you will receive Total Value of Shares
Held for two
 * companies.  But notice that the hb.run you added does not look like other
hb.runs, they all
 * have two parameters whereas yours only has one:
 * hb.run('HB_AID=ENTER', 'T002'); vs hb.run('HB_AID=PF3');
 *
 * The T002 above is the BMS map name for that screen.  The hb.run is checking to
make sure you
 * ended up on the screen you expected to be on after the command was
executed.  The statement
 * hb.run('HB_AID=ENTER', 'T002') says that after pressing the enter I expect to
be on screen T002
 * and if not then throw an error.
 *
 * How do you know what the BMS map names are?  They are displayed in the lower
left corner
 * of the application explorer.  Update the two statements you added with a
second parameter
 * that is the BMS map name the application should be on after pressing the PF3
key.
 * hb.run('HB_AID=PF3', 'BOB');  You can get the BMS map name from the
Application Explorer.  If
 * you get it wrong the error message will tell you the BMS map name that was
encountered and
 * you can correct your code and retest.
 */

function initialize() {
    hb.set('hb_entry', 'trad');
    hb.run('HB_AID=ENTER', 'T002');
}


function getQuote(companyNumber) {
    hb.set('OPTION', companyNumber);
    hb.run('HB_AID=ENTER', 'T003');

    hb.set('OPT2', '1');
    hb.run('HB_AID=ENTER', 'T004');

    writeln('Company Name: ', hb.getFieldValue('COMP41'));
    writeln('Total Value of Shares Held = ', hb.getFieldValue('VALUE'));
    writeln();
```

```
    hb.run('HB_AID=PF3');
    hb.run('HB_AID=PF3');


}

initialize();
getQuote('1');
getQuote('2');
```

Workshop #4

In this workshop you will be adding additional fields to the previous script. The fields that are going to be added are the following:

Share Value Now

Number of Shares Held

Commission Cost for Selling

Commission Cost for Buying

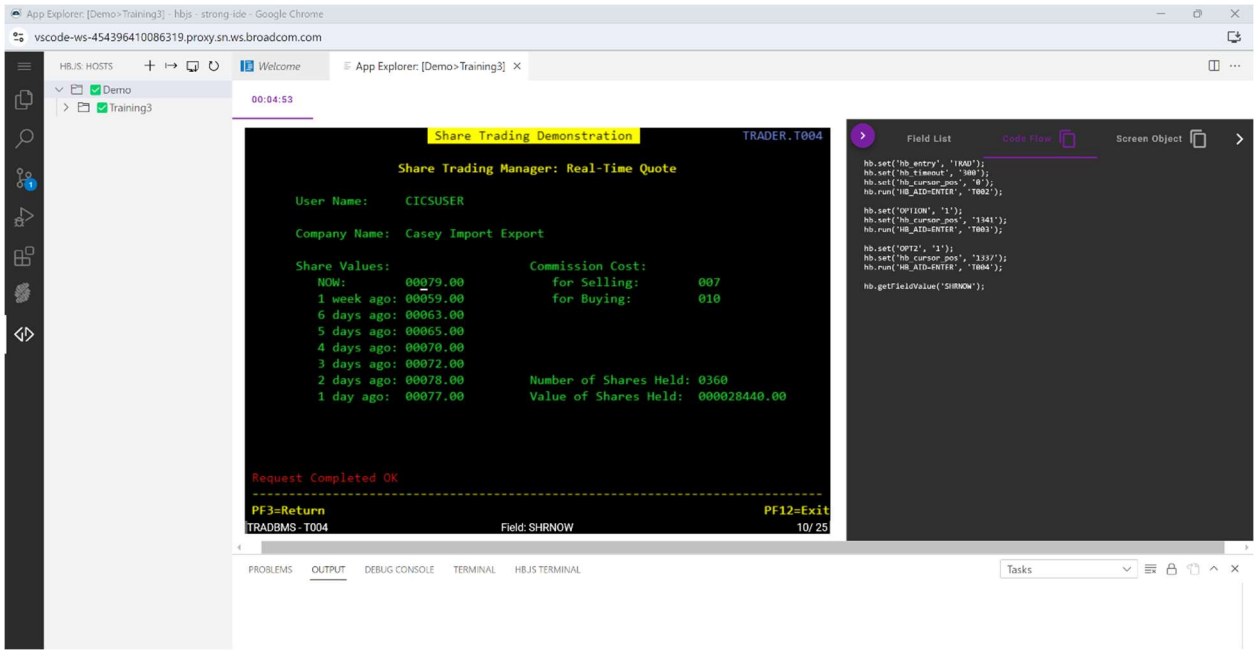
In addition, you will need to perform this task for not just the first two stocks but for all four. This can be accomplished in two different ways. One would be to add a loop to the logic and the other is to hard code it.

You may need to review some of the Broadcom documentations to get explanations on some of the HostBridge utilities that you have been using. Below is a link to the documentation.

https://techdocs.broadcom.com/us/en/ca-mainframe-software/devops/hostbridge-javascript-engine/8-0/using/transaction-directives.html

You will need to access the Application Explorer to gather the information needed for this exercise. Navigate to the Share Trading Demonstration Screen, and place your curser on the value for NOW:

Double click the value.



Notice in the Code Flow the following line was added.

```
hb.getFieldValue('SHRNOW');
```

The take away is the value SHRNOW, this value name will be needed for the exercise. Repeat this process for the remainder of the values. Remember to take only the field values and not the field names.

Return to the script and follow the instructions to complete the workshop.

Below is the hard coded way to complete the workshop.

```
/*
 * Rename this file by right clicking the file name in the explorer to the right
and selecting rename.  The
 * new file name should be prefixed by your user id.  If your user id is
mfwsuser## your new file name would
 * be mfwsuser##_getAllQuotes.hbx
 */

let hb = new HB.Session();

let common = require('common', 'hbutils');
let debugging = require('debugging', 'hbutils');

debugging.checkDebugControl();
common.hbRunProto();

/*
 * Before making any changes run the service (right-click HB.js Commands > Run)
and view the output in
 * the HBJS TERMINAL OUTPUT tab.
 *
 * Below is most of the code you should have had in the last step.  I've added a
main and
 * terminate function to better separate and identify sections of code.  I've
also added
 * a "response" global variable that I am going to use to build up the data I
want to return
 * to the calling client.
 */

/*
 * Here I am declaring "response" as a global array variable.  It is global
because it is
 * declared outside of any function.
 */
var response = new Array();

function initialize() {
    hb.set('hb_entry', 'trad');
    hb.run('HB_AID=ENTER', 'T002');
}
```

```
function terminate() {
    /*
     * PF12 exits the trader application no matter what screen you are on, the
hb_delete_session=1
     * deletes the remaining HB.js resources leaving nothing on the host.  The
session would automatically
     * timeout after 5 minutes of inactivity, but in a highly active system that
might leave thousands
     * or 10's of thousands of unused sessions consuming resources until they are
automatically deleted.
     * You should always clean up your session at the conclusion or your
service.  hb_delete_session along
     * with all HB.js are documented here:
     * https://techdocs.broadcom.com/us/en/ca-mainframe-
software/devops/hostbridge-javascript-engine/8-0.html
     */
    hb.run('hb_aid=pf12&hb_delete_session=1');

    /*
     * Convert response to JSON and write it out
     */
    write(common.toJSONString(response));
}


function getQuote(companyNumber) {
    hb.set('OPTION', companyNumber);
    hb.run('HB_AID=ENTER', 'T003');

    hb.set('OPT2', '1');
    hb.run('HB_AID=ENTER', 'T004');

    // writeln('Company Name: ', hb.getFieldValue('COMP41'));
    // writeln('Total Value of Shares Held = ', hb.getFieldValue('VALUE'));
    // writeln();

    /*
     * Instead of writing anything out here I'm going to put my
     * output into a JavaScript object and let my "terminate" function
     * produce JSON output.
     */

    let quote = {
        companyName: hb.getFieldValue('COMP41'),
```

```
        totalValue: hb.getFieldValue('VALUE'),
        currentPrice: hb.getFieldValue('SHRNOW'),
        shareHeld: hb.getFieldValue('HELD'),
        buyCost: hb.getFieldValue('BUY'),
        sellCost: hb.getFieldValue('SELL)')
    }

    /*
     * Using the Application Explorer and line 75 above as an example add the
following
     * fields from the "Real-Time Quote" to the JavaScript object "quote" above:
     *
     *      Current Share Price
     *      Number of Shares Held
     *      Commission Price for buying and selling shares
     *
     * After adding the fields retest your service (right-click HB.js Run)
     */

    /*
     * Add quote to the end of the response array
     */
    response.push(quote);

    hb.run('hb_aid=pf3', 'T003');
    hb.run('hb_aid=pf3', 'T002');

}

/*
 * Update the main function below to get all four companies.  You could do that
simply by
 * adding to more calls to getQuote, but it would be better to use a for-loop to
iterate over
 * the four companies.  It is easy to find the syntax and examples of JavaScript,
search for
 * "JavaScript for loop".  After updating, run the service (right-click HB.js
Commands > Run)
 */

function main () {
    initialize();
    getQuote('1');
    getQuote('2');
    getQuote('3');
```

```
    getQuote('4');
    terminate();
}

main();
```

Below is the loop way to complete the workshop.

```
/*
 * Rename this file by right clicking the file name in the explorer to the right
and selecting rename.  The
 * new file name should be prefixed by your user id.  If your user id is
mfwsuser## your new file name would
 * be mfwsuser##_getAllQuotes.hbx
 */

let hb = new HB.Session();

let common = require('common', 'hbutils');
let debugging = require('debugging', 'hbutils');

debugging.checkDebugControl();
common.hbRunProto();

/*
 * Before making any changes run the service (right-click HB.js Commands > Run)
and view the output in
 * the HBJS TERMINAL OUTPUT tab.
 *
 * Below is most of the code you should have had in the last step.  I've added a
main and
 * terminate function to better separate and identify sections of code.  I've
also added
 * a "response" global variable that I am going to use to build up the data I
want to return
 * to the calling client.
 */

/*
 * Here I am declaring "response" as a global array variable.  It is global
because it is
 * declared outside of any function.
```

```
 */
var response = new Array();

function initialize() {
    hb.set('hb_entry', 'trad');
    hb.run('HB_AID=ENTER', 'T002');
}

function terminate() {
    /*
     * PF12 exits the trader application no matter what screen you are on, the
hb_delete_session=1
     * deletes the remaining HB.js resources leaving nothing on the host.  The
session would automatically
     * timeout after 5 minutes of inactivity, but in a highly active system that
might leave thousands
     * or 10's of thousands of unused sessions consuming resources until they are
automatically deleted.
     * You should always clean up your session at the conclusion or your
service.  hb_delete_session along
     * with all HB.js are documented here:
     * https://techdocs.broadcom.com/us/en/ca-mainframe-
software/devops/hostbridge-javascript-engine/8-0.html
     */
    hb.run('hb_aid=pf12&hb_delete_session=1');

    /*
     * Convert response to JSON and write it out
     */
    write(common.toJSONString(response));
}


function getQuote(companyNumber) {
    hb.set('OPTION', companyNumber);
    hb.run('HB_AID=ENTER', 'T003');

    hb.set('OPT2', '1');
    hb.run('HB_AID=ENTER', 'T004');

    // writeln('Company Name: ', hb.getFieldValue('COMP41'));
    // writeln('Total Value of Shares Held = ', hb.getFieldValue('VALUE'));
    // writeln();

    /*
```

```
     * Instead of writing anything out here I'm going to put my
     * output into a JavaScript object and let my "terminate" function
     * produce JSON output.
     */

    let quote = {
        companyName: hb.getFieldValue('COMP41'),
        totalValue: hb.getFieldValue('VALUE'),
        currentPrice: hb.getFieldValue('SHRNOW'),
        shareHeld: hb.getFieldValue('HELD'),
        buyCost: hb.getFieldValue('BUY'),
        sellCost: hb.getFieldValue('SELL)')
    }

    /*
     * Using the Application Explorer and line 75 above as an example add the
following
     * fields from the "Real-Time Quote" to the JavaScript object "quote" above:
     *
     *      Current Share Price
     *      Number of Shares Held
     *      Commission Price for buying and selling shares
     *
     * After adding the fields retest your service (right-click HB.js Run)
     */

    /*
     * Add quote to the end of the response array
     */
    response.push(quote);

    hb.run('hb_aid=pf3', 'T003');
    hb.run('hb_aid=pf3', 'T002');

}

/*
 * Update the main function below to get all four companies.  You could do that
simply by
 * adding to more calls to getQuote, but it would be better to use a for-loop to
iterate over
 * the four companies.  It is easy to find the syntax and examples of JavaScript,
search for
 * "JavaScript for loop".  After updating, run the service (right-click HB.js
Commands > Run)
```

```
 */

function main () {
    initialize();
    for(let i = 1; i <= 4; ++i){
        getQuote(i);
    }
    terminate();
}


main();
```

Workshop #5

In the last exercise you were able to test all 4 companies. In this exercise you will only test companies 1 and 3 and also you will be using the web browser so you will lose the ability to test in VS Code. Follow the comments in 5_getSelectedQuotes this will have you modify the code to add the ability to select the number of companies you want to test.

Below is the modified code to complete the workshop.

```
/*
 * Rename this file by right clicking the file name in the explorer to the right
and selecting rename.  The
 * new file name should be prefixed by your user id.  If your user id is
mfwsuser## your new file name would
 * be mfwsuser##_getSelectedQuotes.hbx
 */

let hb = new HB.Session();

let common = require('common', 'hbutils');
let debugging = require('debugging', 'hbutils');

debugging.checkDebugControl();
common.hbRunProto();

/*
 * Below is most of the code you should have had at the end of 4-getAllQuotes.
This service gets
 * all company quotes, but what if you only want the quote for company 1 or for
companies 1 and 3,
 * or for some other combination.  We should be able to pass a parameter to the
service to specifiy
 * what companies we want the real-time quote for instead of always returning all
companies.
```

```
 * We will do that by passing a parameter on the query string to specifiy what
companies we want
 * quotes.  Let's get our input in the initialize function.
 */

var response = new Array();

function initialize() {
    hb.set('hb_entry', 'trad');
    hb.run('HB_AID=ENTER', 'T002');

    /*
     * Now that we are passing input in via the query string we can no longer
test
     * our service by executing it in VS Code.  From now on we will have to
"make" or
     * deploy our service to test it, if you right-click HB.js Commands > Run the
service will fail
     * because there will be no value for companies
     */



    let companies = HB.request.http.getValue('companies');

    /*
     * The value of companies above will be a string containing "1,3".  We need
to be able to
     * iterate over the company numbers.  Luckily there is a method called
"split" that hangs off
     * all strings that can split a string into an array based on a separator.
     * companies.split(',') will separate the company numbers into array elements
allowing us to
     * easliy interate over them.
     *
     * We could put companies in a global variable, but using too many global
variables is considered
     * poor programming, instead we will split the string into an array and pass
it
     * back to the main function.
     */

    return companies.split(',');
}
```

```
function terminate() {

    hb.run('hb_aid=pf12&hb_delete_session=1');

    /*
     * Below I added an http content type header to indicated the output is JSON
     */
    common.headers.json();

    write(common.toJSONString(response));
}


function getQuote(companyNumber) {
    hb.set('OPTION', companyNumber);
    hb.run('HB_AID=ENTER', 'T003');

    hb.set('OPT2', '1');
    hb.run('HB_AID=ENTER', 'T004');

    let quote = {
        companyName: hb.getFieldValue('COMP41'),
        currentPrice: hb.getFieldValue('SHRNOW'),
        sharesHeld: hb.getFieldValue('HELD'),
        buyCommission: hb.getFieldValue('BUY'),
        sellCommission: hb.getFieldValue('Sell'),
        totalValue: hb.getFieldValue('VALUE')
    }

    response.push(quote);

    hb.run('hb_aid=pf3', 'T003');
    hb.run('hb_aid=pf3', 'T002');

}


function main () {
    /*
     * Initialize was updated to return a list of companies we are to
     * get the real-time quote for, so here we must catch that array of
     * companies that are returned
     */
    let companies = initialize();
```

```
    /*
     * Below is our original loop.  Updated it to iterate over the list of
     * companies.  Again there are lots of examples of JavaScript for loops
     * iterating over a list of array elements on the net.  After updating
     * the loop below "make it" (right-click HB.js Commands > Make) and then run
it from a
     * browser
     */

    for (let i = 0; i < companies.length; i++){

        getQuote(companies[i].toString());
    }

    /*
     * You can compile and deploy (MAKE) this code by right-clicking in this
editor window and selecting
     * "HB.js Commands > Make". You can invoke the deployed web service by using
a web browser and opening the link
     * below.  Be sure and change the service name to the name you used
above.  Do not include the ".hbx"
     *
     * https://external-
943025290693663.proxy.sn.ws.broadcom.com/hbscript/mfwsuser##_getSelectedQuotes?com
panies=1,3
     */

    terminate();
}

main();
```

Workshop #6

Congratulations you have made it to the last workshop. This is a workshop we are going to walk through together and explain the new commands and ideas on how to code the script. Some of the new commands are Try Catch which will help protect you from errors and to ensure you have values returned. What if the users want the choice of XML or JSON? What if you want more statistics?

For those who are adventurous at the very bottom is an option set of instructions to add more error checking. If you're up to the challenge they to code the changes.

```
/*
 * Rename this file by right clicking the file name in the explorer to the right
and selecting rename.  The
 * new file name should be prefixed by your user id.  If your user id is
mfwsuser## your new file name would
 * be mfwsuser##_getQuoteFinal.hbx
 */

/*
 * Below is a near production ready web service.  Added are try catches to
protect from errors and to be sure
 * to always return valid JSON.  Also the option to return XML instead of JSON
was added along with more
 * information on how the service ran, elapsed time, number of transaction
executed, etc
 */

// If I am going to measure the elapsed time I need to capture the start time of
the service in microseconds
let startTime = HB.stckf();
```

```
let hb = new HB.Session();

let common = require('common', 'hbutils');
let debugging = require('debugging', 'hbutils');

debugging.checkDebugControl();
common.hbRunProto();

/*
 * Lets update response to return more information than just the real-time quote
 */
var response = {
    quotes: [],             // quotes will still contain the real time quote for
each company
    status: {               // Create a status block to keep all the info about
how the service ran
        success: false,     // Create a single flag the client can check to see
if the service ran okay
        elapsedTime: 0,     // How long did it take for the service to run,
elapsed time
        tranCount: 0,       // How many transactions did we run
        errors: []          // A place to store error messages and the error
stack
    }
}

function initialize() {
    hb.set('hb_entry', 'trad');
    hb.run('HB_AID=ENTER', 'T002');

    let companies = HB.request.http.getValue('companies');

    if (typeof(companies) === 'undefined') {
        companies = '1,2,3,4';      // If companies is undefined then lets return
all company real-time quotes
    }

    let type = HB.request.http.getValue('type');

    if (typeof(type) === 'undefined') {
        type = 'json'               // If type is undefined then we will return
JSON output
    }

    return ({
```

```
        companies: companies.split(','),
        type : type
    });
}

function terminate(type) {

    hb.run('hb_aid=pf12&hb_delete_session=1');

    /*
     * Now we are at the end lets set the elapsed time and transaction count into
the response
     */
    response.status.elapsedTime = HB.stckf() - startTime;
    response.status.tranCount = hb.tranCount;

    /*
     * Do we output XML or JSON, type will know
     */
    if (type === 'xml') {
        common.headers.xml();
        write(common.jsToXML(response, 'getQuote'));
    } else {
        common.headers.json();
        write(common.toJSONString(response));
    }
}


function getQuote(companyNumber) {
    hb.set('OPTION', companyNumber);
    hb.run('HB_AID=ENTER', 'T003');

    hb.set('OPT2', '1');
    hb.run('HB_AID=ENTER', 'T004');

    let quote = {
        companyName: hb.getFieldValue('COMP41'),
        currentPrice: hb.getFieldValue('SHRNOW'),
        sharesHeld: hb.getFieldValue('HELD'),
        buyCommission: hb.getFieldValue('BUY'),
        sellCommission: hb.getFieldValue('Sell'),
        totalValue: hb.getFieldValue('VALUE')
    }
```

```
    // We now want to push are individual quotes on to the quotes array in the
response
    response.quotes.push(quote);

    hb.run('hb_aid=pf3', 'T003');
    hb.run('hb_aid=pf3', 'T002');

}


function main () {

    /*
     * We need to wrapper all of our code in a try catch so that if
     * any errors occur we will always output either JSON or XML
     */
    try {
        /*
        * Initialize not only returns the list of companies to be processed
        * but also the type of output to be produced
        */
        var input = initialize();

        for (let i = 0; i < input.companies.length; i++) {
            getQuote(input.companies[i]);
        }

        // if we get to here then no errors occured and we can indicate that
        // the service was a success
        response.status.success = true;

    } catch (e) {
        // If an error was caught then push it on to the error array so that it
        // can be produced in the JSON or XML output
        response.status.errors.push(e);
    } finally {
        // If we run successfully or not we always want to run terminate
        terminate(input.type);
    }
}



main();
```

```
/*
 * You can compile and deploy (MAKE) this code by right-clicking in this editor window and selecting
 * "HB.js Commands > Make". You can invoke the deployed web service by using a web browser and opening the link
 * below.  Be sure and change the service name to the name you used above.  Do not include the ".hbx"
 *
 * https://external-943025290693663.proxy.sn.ws.broadcom.com/hbscript/mfwsuser##_getQuoteFinal?companies=1,3&type=xml
 */


/*
 * More protections could be built into this service.  Try running the service with companies=X or companies=6.
 * Can you improve this service further???
 */
```