HCA Tech Note 130

Object handles in programs

When you use the JsonOpen, DesignOpen, and FileOpen functions in the Compute element of a program, you get back what is called a “handle”. This is a number (between 0 and 16 but you shouldn’t care what the value is) that is used for Json, design, or file operations. It is also the value supplied to the JsonClose, DesignClose, or FileClose functions when you are done. You had to make sure that you used the close function as there are a limited number of these objects that can be opened at once. If you didn’t use the close function, the object would still be marked as open and eventually you would run out of that limited number and subsequent JsonOpen, DesignOpen, and FileOpen functions would fail. While you should still explicitly close any handles you open, in some situations HCA will close them for you if it finds that you have not.

NOTE: If you don’t use these any of these three functions in Compute elements then stop reading now unless you are interested in this advanced feature.

During program execution there is a stack that keeps track of execution state including local variables in what is called a “frame”. If program A starts program B, and program B starts program C, the stack has 3 items on it when executing C with the frame for C at the top of the stack.

When a program C finishes execution, control returns to the program B, and the frame for program C is removed from the stack. Before the frame for the program ending is removed, HCA looks at the local variables in that frame and if any were for an open design, file, or json object, that object gets closed if the handle is held in a local variable. You still should use the appropriate close function in your program but if you forget, now it gets closed for you.

There are other places where this ”automatic close” happens too.

1. If program X is stopped by the UI or from another program doing a “Stop” element on program X, then any object that X has open gets closed as part of stopping X. HCA looks at all existing stack frames for the program when it is stopped and finds local variables that hold object handles to close.

2. If program X makes an error in a Compute element and the remainder of the Compute element is skipped, then a handle could have been left open. Like this:

```
hDesign = _DesignOpen(1);
aDate   = _Date(2019, 80, 79);  // Generates a program error
void    = _DesignClose(hDesign);
```

HCA never gets to execute the DesignClose as the Date function is in error. Because of that, the handle is left open. HCA closes it, if hDesign is a local variable, when the program finishes.

3. In the debugger if your program hits a breakpoint and you start it in the debugger and close the debugger window before the program completes, objects could have been left open. Those are now closed as part of the debugger closing.

Because HCA programs use untyped variables that take on any value and type as needed, HCA now has a hidden type associated with a variable that contains a handle. It still looks like a number and works the same as before. This works:

```
hDesign =_designOpen(1);
x       = hDesign;
name    = _DesignName(x);
```

hDesign is internally identified as a handle, x is just a number, but the DesignName function doesn't care. It's the value of the argument is all that matters.

If you mess with handles, they could lose their "handle-ness".

```
hDesign = _DesignOpen(1);
hDesign = hDesign + 1 - 1;
```

Then hDesign wouldn't be a design handle anymore even though it still had the same value. If you did this then HCA wouldn't know that hDesign contained a design handle so it wouldn't automatically close it if you left it open. (But this is a silly example, and no one does this, right?)

If you pass a handle to another program, it "handleness" is lost. That's a good thing! When the invoked program finishes and returns to the invoking program it doesn't look like a handle so isn't closed.

All of this doesn't apply if you use a Global variable. You are responsible to close them.

The good news is that the debugger lets you see what handles are open and manually close them. If, for example, the handle is held in a global variable and the program is stopped with the handle open, just debug any program and use the handle explorer to view and close it.

##end##