Scripting Concepts & Debugging

Class Description

The Advanced Scripting class is designed to familiarize students advanced concepts about the powerful scripting engine embedded within Accela Automation. This class assumes that the administrator has a basic understanding of Accela Automation and EMSE scripting.

Class Length

The Advanced Scripting class is designed for 2 hours of instruction.

Instructional Delivery Methods

Instructor led, hands-on exercises.

Contents

Exercise 1 – Installing a 3.0 Master Script

Objective:

We will learn how to install and test the 3.0 master scripts.   We'll demonstrate the backward compatibility of the scripts by testing existing business rules.

Exercise:

1. Located the Enterprise Scripts 3.0 zipped distribution located Libraries\Documents\Master Scripts:

2. Unzip the distribution zip file to its own directory (right click, 7-Zip, extract to "Enterprise Scripts…"

3. Create a data manager job to install the master script distribution zip file (Located in the Misc folder)

4. Verify that the scripts are loaded.   How?

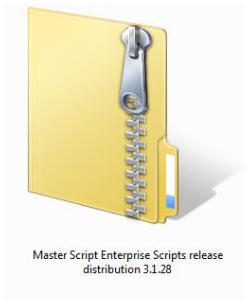
5. Assign the ApplicationSpecificInfoUpdateAfter event to its corresponding 3.0 master script.

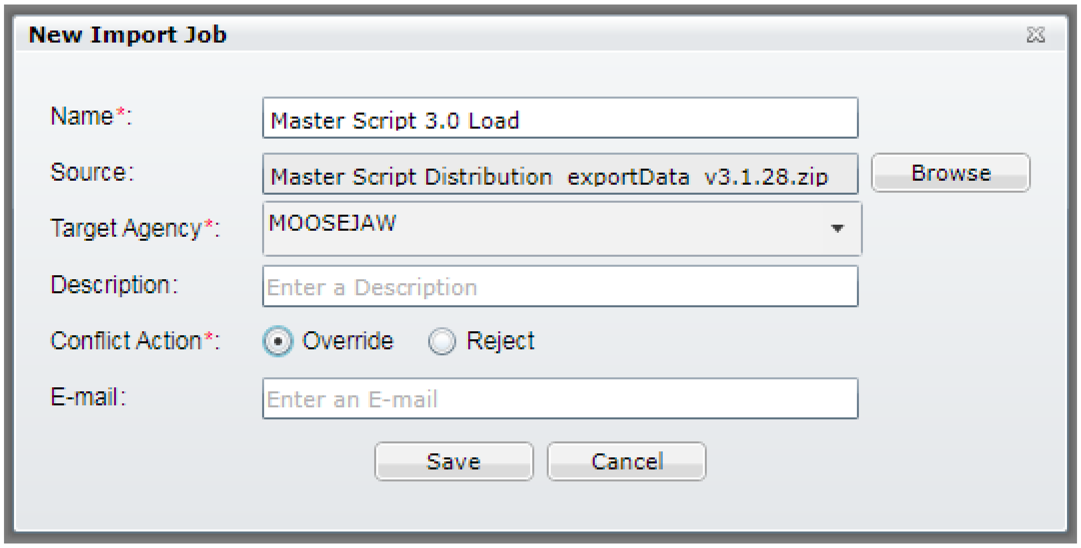
6. Trigger the ASIUA event.

   Notes and Discussion:

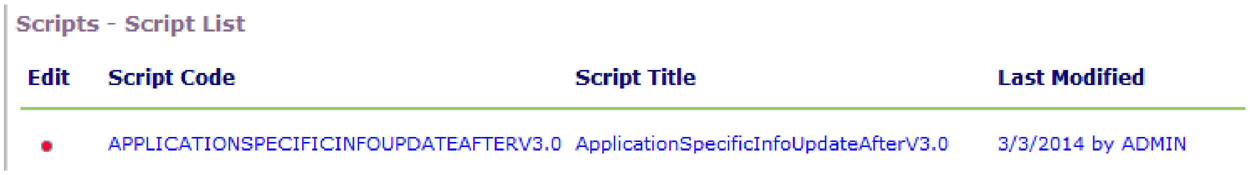
   - What differences can you see in the actual script execution?
   - Did any standard choices execute?   Why?

Exercise 1 – Key

1. 
Master Script Enterprise Scripts release distribution 3.1.28

3. 
New Import Job

Name*:	Master Script 3.0 Load
Source:	Master Script Distribution exportData v3.1.28.zip
Target Agency*:	MOOSEJAW
Description:	Enter a Description
Conflict Action*:	[✓ Override] Reject
E-mail:	Enter an E-mail

4. 
Scripts - Script List

Edit	Script Code	Script Title	Last Modified
●	APPLICATIONSPECIFICINFOUPDATEAFTERV3.0	ApplicationSpecificInfoUpdateAfterV3.0	3/3/2014 by ADMIN

Exercise 2 – Tie Events to Scripts

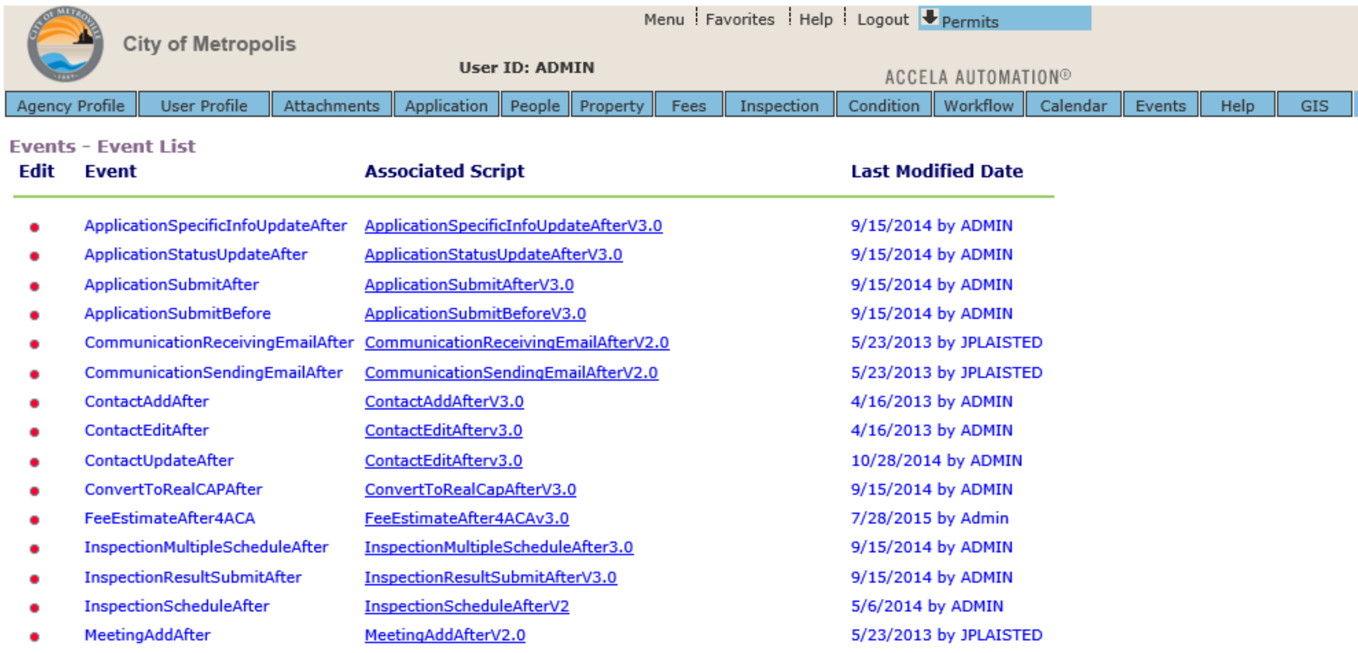
Objective:

Tie your Events to Master Scripts 3.0

Exercise:

1. In Classic AA -> Events -> Events, add the following events.
    a. **ApplicationSpecificInfoUpdateAfter**
    b. **ApplicationSubmitAfter**
    c. **WorkflowTaskUpdateAfter**
2. Clear Cache

Exercise 2 – Key


City of Metropolis
Menu | Favorites | Help | Logout | Permits
User ID: ADMIN
ACCELA AUTOMATION®
Agency Profile | User Profile | Attachments | Application | People | Property | Fees | Inspection | Condition | Workflow | Calendar | Events | Help | GIS

Events - Event List

Edit	Event	Associated Script	Last Modified Date
●	ApplicationSpecificInfoUpdateAfter	ApplicationSpecificInfoUpdateAfterV3.0	9/15/2014 by ADMIN
●	ApplicationStatusUpdateAfter	ApplicationStatusUpdateAfterV3.0	9/15/2014 by ADMIN
●	ApplicationSubmitAfter	ApplicationSubmitAfterV3.0	9/15/2014 by ADMIN
●	ApplicationSubmitBefore	ApplicationSubmitBeforeV3.0	9/15/2014 by ADMIN
●	CommunicationReceivingEmailAfter	CommunicationReceivingEmailAfterV2.0	5/23/2013 by JPLAISTED
●	CommunicationSendingEmailAfter	CommunicationSendingEmailAfterV2.0	5/23/2013 by JPLAISTED
●	ContactAddAfter	ContactAddAfterV3.0	4/16/2013 by ADMIN
●	ContactEditAfter	ContactEditAfterv3.0	4/16/2013 by ADMIN
●	ContactUpdateAfter	ContactEditAfterv3.0	10/28/2014 by ADMIN
●	ConvertToRealCAPAfter	ConvertToRealCapAfterV3.0	9/15/2014 by ADMIN
●	FeeEstimateAfter4ACA	FeeEstimateAfter4ACAv3.0	7/28/2015 by Admin
●	InspectionMultipleScheduleAfter	InspectionMultipleScheduleAfter3.0	9/15/2014 by ADMIN
●	InspectionResultSubmitAfter	InspectionResultSubmitAfterV3.0	9/15/2014 by ADMIN
●	InspectionScheduleAfter	InspectionScheduleAfterV2	5/6/2014 by ADMIN
●	MeetingAddAfter	MeetingAddAfterV2.0	5/23/2013 by JPLAISTED

1.

Exercise 3 – Developing with Script Tester

Objective:

In this exercise we'll be trying out the Script Tester, which allows a developer to code scripts outside of standard choices for the purposes of testing.

Exercise:

1. Locate the *ScriptTest.js* file on your machine. Open it in notepad and paste it into script test. At the top of the script, replace the existing Alt-ID with an Alt-ID from your current agency.

2. Immediately after the "Add User Code" comment, add code to create a new inspection on the record. Make sure you enclose this code in a try/catch block to ensure that any errors are caught.

3. Submit the script. Go to the Inspection portlet and see if the new inspection has been created. If not, why not?

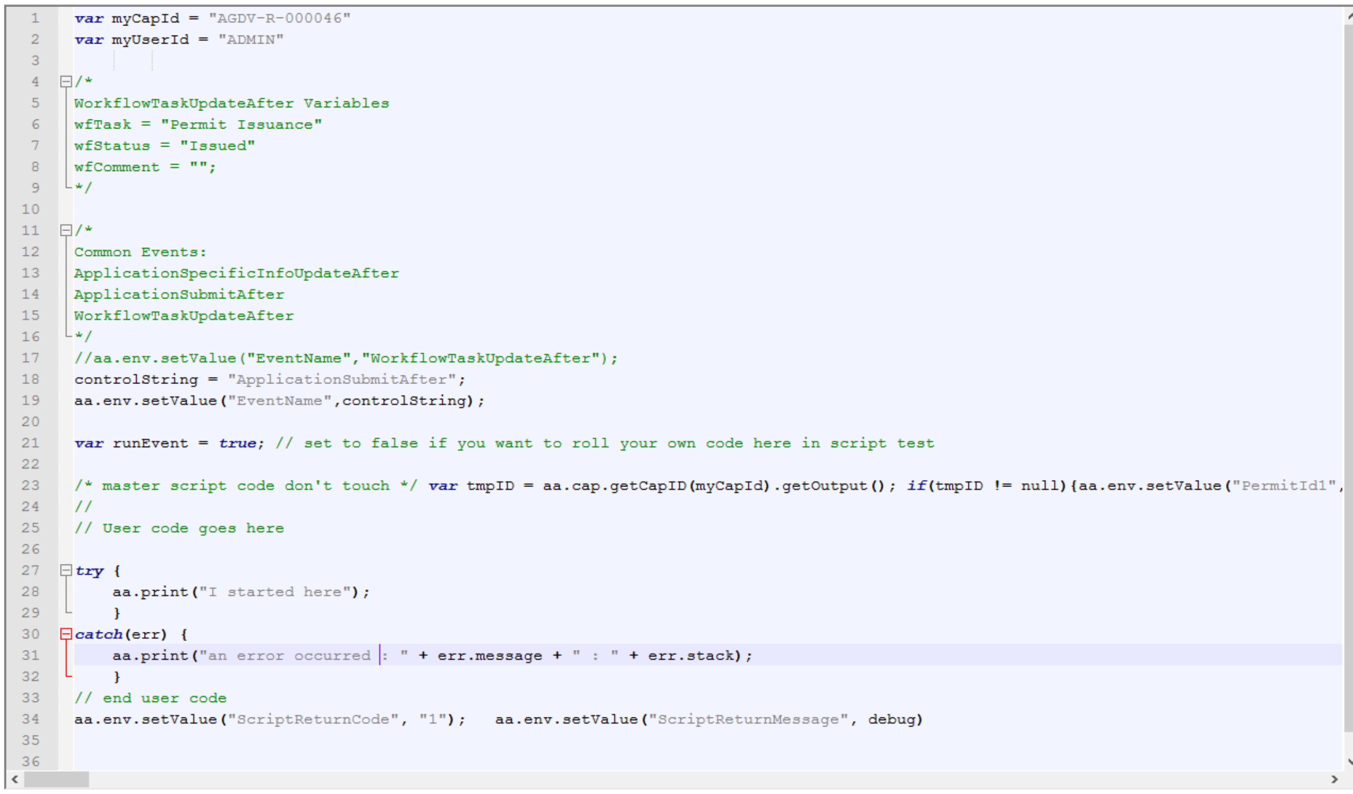
4. Next, review the master script function list. Pick one or more functions that you're interested in to perform actions on this record. For example, you could update workflow, add a condition, add fees, create a child record, etc.

5. Update your test script by adding these calls after the "Add User Code" comment. Enclose in a try/catch block to trap any errors. Have fun!

Notes and Discussion:

- Describe the differences and similarities between this Script Tester and a Master Script.
- Do you see any similarities between this script and the one we created in exercises 1-4?
- What's the difference between "Always Rollback", "Commit if Successful", and "Use User Transaction" options?
- What are sets useful for?
- What is the ScriptReturnCode used for?
    - Hint: See Understanding Script Return Values in the Accela Automation 7.3 FP3 Scripting Guide

Exercise 3 – Key

1. 

```
var myCapId = "AGDV-R-000046"
var myUserId = "ADMIN"

/*
WorkflowTaskUpdateAfter Variables
wfTask = "Permit Issuance"
wfStatus = "Issued"
wfComment = "";
*/

/*
Common Events:
ApplicationSpecificInfoUpdateAfter
ApplicationSubmitAfter
WorkflowTaskUpdateAfter
*/
//aa.env.setValue("EventName","WorkflowTaskUpdateAfter");
controlString = "ApplicationSubmitAfter";
aa.env.setValue("EventName",controlString);

var runEvent = true; // set to false if you want to roll your own code here in script test

/* master script code don't touch */ var tmpID = aa.cap.getCapID(myCapId).getOutput(); if(tmpID != null){aa.env.setValue("PermitId1",
//
// User code goes here

try {
    aa.print("I started here");
    }
catch(err) {
    aa.print("an error occurred : " + err.message + " : " + err.stack);
    }
// end user code
aa.env.setValue("ScriptReturnCode", "1");   aa.env.setValue("ScriptReturnMessage", debug)


```

Exercise 4 – Add a Fee Business Script

Objective:
Using the script tester, we will create a business script to add a fee to the record on ApplicationSpecificInfoUpdateAfter

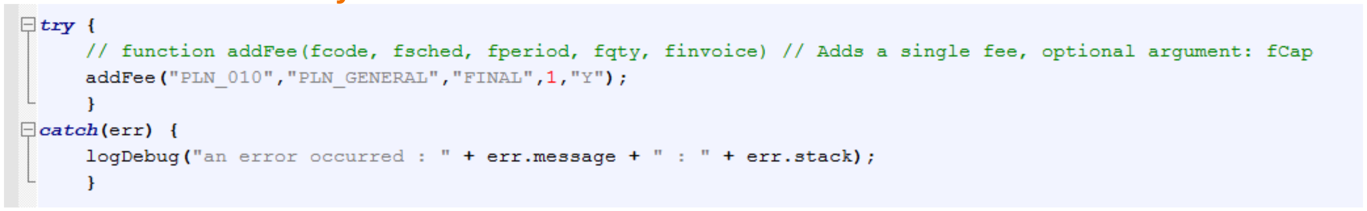
Exercise:

1. Lookup the addFee function in the Scripting Guide
2. Find a fee under Fee Schedules
3. Write the code necessary to add the fee to your record

Notes and Discussion:

- Make sure you have a fee that exists in your environment or the script will not work.

Exercise 4 – Key



```
try {
    // function addFee(fcode, fsched, fperiod, fqty, finvoice) // Adds a single fee, optional argument: fCap
    addFee("PLN_010","PLN_GENERAL","FINAL",1,"Y");
    }
catch(err) {
    logDebug("an error occurred : " + err.message + " : " + err.stack);
    }
```

Exercise 5 – Schedule an Inspection

Objective:
Using the script tester, we will create a business script to schedule an inspection to the record on WorkflowTaskUpdateAfter

Exercise:

5a

1. Lookup the scheduleInspection function in the Scripting Guide
2. Find an inspection type under Inspections
3. Write the code necessary to schedule to your record

5b

1. Add a workflow task & status to your script

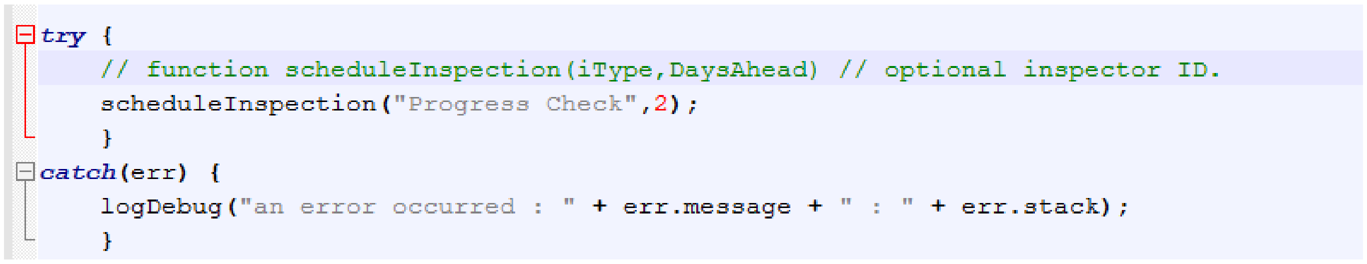
   Notes and Discussion:

   - What other functions exist in the Master Scripts for scheduling inspections?
   - We no longer need to set the workflow from the User Interface to test a particular task and status because it is accomplished within the Script Tester

Accela

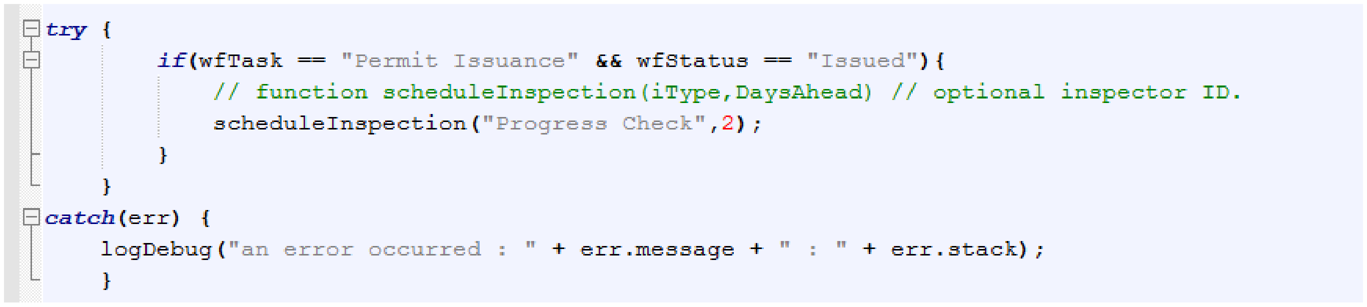
Exercise 5 – Key

5a



```
try {
    // function scheduleInspection(iType,DaysAhead) // optional inspector ID.
    scheduleInspection("Progress Check",2);
    }
catch(err) {
    logDebug("an error occurred : " + err.message + " : " + err.stack);
    }
```

5b



```
try {
        if(wfTask == "Permit Issuance" && wfStatus == "Issued"){
            // function scheduleInspection(iType,DaysAhead) // optional inspector ID.
            scheduleInspection("Progress Check",2);
        }
    }
catch(err) {
    logDebug("an error occurred : " + err.message + " : " + err.stack);
    }
```

Exercise 6 – Create a Child Record

Objective:
Using the script tester, we will create a business script to create a child to the record on WorkflowTaskUpdateAfter

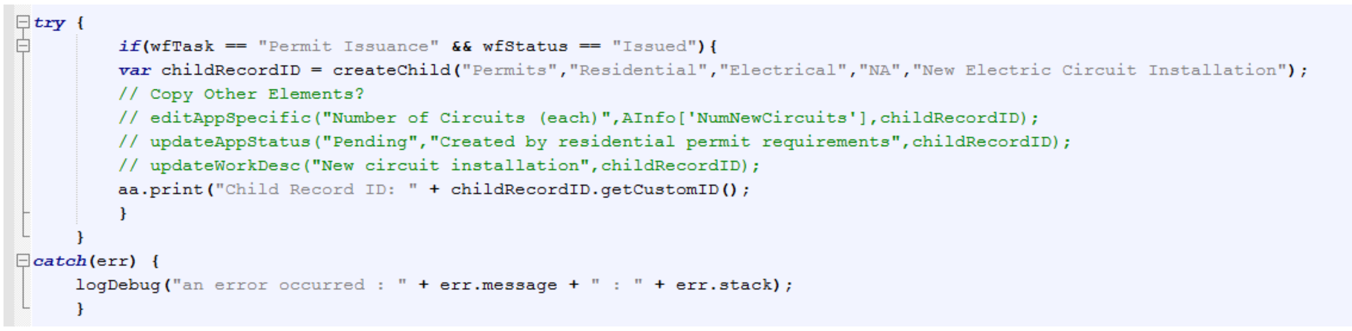
Exercise:

1. Lookup the createChild function in the Scripting Guide
2. Find a child record type to create
3. Write the code necessary to create a child record

Notes and Discussion:

- What specifically gets copied from the parent to the child?
- What if don’t want all that stuff copied?!?

Exercise 6 – Key



```
try {
        if(wfTask == "Permit Issuance" && wfStatus == "Issued"){
        var childRecordID = createChild("Permits","Residential","Electrical","NA","New Electric Circuit Installation");
        // Copy Other Elements?
        // editAppSpecific("Number of Circuits (each)",AInfo['NumNewCircuits'],childRecordID);
        // updateAppStatus("Pending","Created by residential permit requirements",childRecordID);
        // updateWorkDesc("New circuit installation",childRecordID);
        aa.print("Child Record ID: " + childRecordID.getCustomID();
        }
    }
catch(err) {
    logDebug("an error occurred : " + err.message + " : " + err.stack);
    }
```

Exercise 7 – Create a Child Record but No Contacts

Objective:
Using the script tester, we will create a business script to create a child to the record on WorkflowTaskUpdateAfter

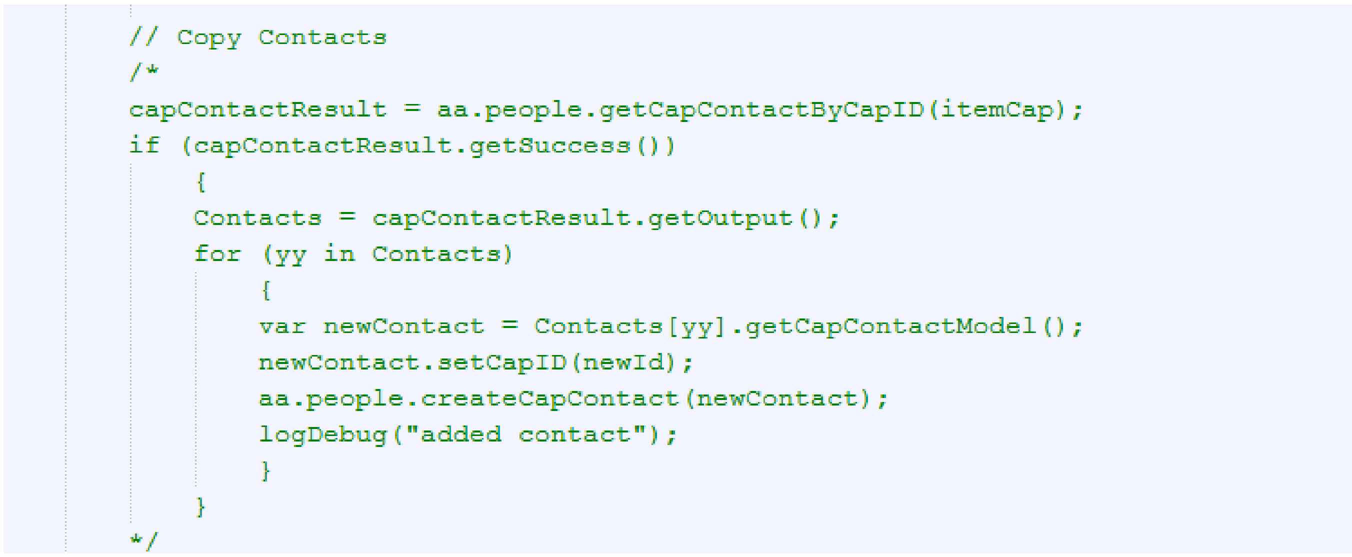
Exercise:

1. Lookup the createChild function in the Scripting Guide
2. Copy the code from it and paste it into your editor
3. Modify the function so that it does not copy contacts

Notes and Discussion:

- What other elements might we want to copy from a parent record to a child?
- What other functions exist in the Master Scripts to copy data elements?

Exercise 7 – Key



```
// Copy Contacts
/*
capContactResult = aa.people.getCapContactByCapID(itemCap);
if (capContactResult.getSuccess())
    {
    Contacts = capContactResult.getOutput();
    for (yy in Contacts)
        {
        var newContact = Contacts[yy].getCapContactModel();
        newContact.setCapID(newId);
        aa.people.createCapContact(newContact);
        logDebug("added contact");
        }
    }
*/
```