**Consultant’s Guide to EMSE 3.0 Conversions**

Contents

[Revision History 4](#_Toc487185971)

[EMSE Upgrade Overview 4](#_Toc487185972)

[Existing EMSE Version 5](#_Toc487185973)

[Upgrading from 2.0 5](#_Toc487185974)

[Upgrading from 1.x 6](#_Toc487185975)

[Upgrading from Hybrid approach 7](#_Toc487185976)

[Existing Product Version 7](#_Toc487185977)

[Productized vs Non-Productized Usage 7](#_Toc487185978)

[Non-Event Script Types 8](#_Toc487185979)

[Batch Scripts 8](#_Toc487185980)

[Set Scripts 9](#_Toc487185981)

[Pageflow Scripts 9](#_Toc487185982)

[Custom Script vs INCLUDES\_CUSTOM 9](#_Toc487185983)

[Tools 9](#_Toc487185984)

[Microsoft Access Conversion Tool Package 9](#_Toc487185985)

[BizDomain Converter 10](#_Toc487185986)

[Source Code Control System Repository 12](#_Toc487185987)

[IDE – Eclipse or Notepad++ with Tortoise SVN 12](#_Toc487185988)

[Agency Repository and Civic Platform Integration 13](#_Toc487185989)

[Process 13](#_Toc487185990)

[Data Collected from the Agency 13](#_Toc487185991)

[Identifying Conversion Risks 14](#_Toc487185992)

[Configuration or EMSE Changes 14](#_Toc487185993)

[Branch Order Execution 14](#_Toc487185994)

[EMSE Standard Choices Not Marked appropriately 14](#_Toc487185995)

[Analysis of 1.x to 3.0 15](#_Toc487185996)

[Step 1: Examine the Master Script Code associated to Active Events 15](#_Toc487185997)

[Step 2: Identify the Control Strings invoked by each Master Script 15](#_Toc487185998)

[Step 3: Identify EMSE Standard Choices (active or not) for Documentation 16](#_Toc487185999)

[Step 4: Identify/Disable Inactive Branches (standard choices) 16](#_Toc487186000)

[Step 5: Create a Conversion Directory (File Structure) 17](#_Toc487186001)

[Step 6a: Import Standard Choices into a Sandbox environment 17](#_Toc487186002)

[Step 7a: Clean Standard Choices 17](#_Toc487186003)

[Step 8a: Run BIZDOMAIN Converter in Script Test 17](#_Toc487186004)

[Step 6b: Prepare the worksheet for JavaScript format conversion 19](#_Toc487186005)

[Step 7b: Convert JSON file to standard choices 19](#_Toc487186006)

[Step 8b: Examine and set EMSE flag for the new standard choices 20](#_Toc487186007)

[Step 9b: Run BIZDOMAIN Converter in Script Test 22](#_Toc487186008)

[Step 10: Identify frequency of branch calls 23](#_Toc487186009)

[Step 11: Omit “Debug only” Standard Choices 24](#_Toc487186010)

[Format and Best Practices 25](#_Toc487186011)

[Debug Control 25](#_Toc487186012)

[Comments 25](#_Toc487186013)

[JavaScript Layout and Indentation 25](#_Toc487186014)

[Elimination of Redundant Code/Custom Functions 25](#_Toc487186015)

[Try/Catch 25](#_Toc487186016)

[Documentation 25](#_Toc487186017)

[Items Omitted from Conversion 25](#_Toc487186018)

[Suggestions for Cleaning Up Code 26](#_Toc487186019)

[Existing Errors in Code 26](#_Toc487186020)

[Unused Events 26](#_Toc487186021)

[Unit Test Notes and Observations 26](#_Toc487186022)

[User Acceptance Testing 26](#_Toc487186023)

# Revision History

| ***Author*** | ***Date*** | ***Revision*** | ***Description*** |
| --- | --- | --- | --- |
| Jeanne Chalk | 6/29/2017 | 1.0 | Initial methodology draft |
|  |  |  |  |

# EMSE Upgrade Overview

As the Event Manager Scripting Engine (EMSE) continues to evolve, many agencies contract with Accela Delivery for assistance with upgrading their current script methodology to the latest 3.0 EMSE approach. These projects typically range 2 to 4 months and are relatively straightforward effort. The number of hours required varies based on the existing EMSE Version and the number and types of scripts employed by the agency at the time of upgrade. These projects are also often paired with the Basic EMSE Training (8 hours without repository training or 12 hours with repository training) so that agencies are adequately prepared to maintain their newly deployed script model and develop new scripts moving forward.

# Existing EMSE Version

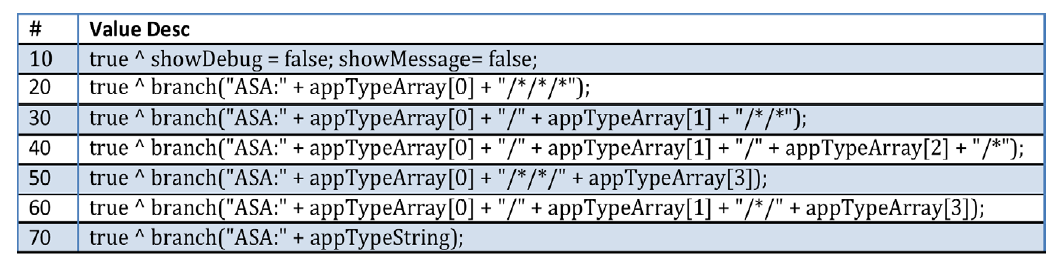
The complexity, risks, and duration of the upgrade varies according to the scripting methodology currently in use.

## Upgrading from 2.0

Many agencies have upgraded to the EMSE 2.0 Methodology and as such, these 3.0 upgrades are far less complex, lower risk, and can be quickly executed and delivered.

The EMSE 2.0 Methodology is characterized by:

* Use of the ***INCLUDES\_....*** library scripts:
  + The master script functions are consolidated into concise script files (***INCLUDES\_ACCELA\_FUNCTIONS & INCLUDES\_ACCELA\_FUNCTIONS\_ASB***)
  + Modifications to master script functions or new custom functions are stored in a single location (***INCLUDES\_CUSTOM***). In more recent deployments, there are also other specialty INCLUDES files for licensing and batch processing.
  + Standard script execution variable defaults are executed at the beginning of each master script via the ***INCLUDES\_ACCELA\_GLOBALS*** file.
* Consistent use of Variable Branching – Variable Branching is a means of executing code specifically related to a record type as appropriate. Each Master Script contains a control string standard choice (a starting point for branch execution). When an agency is consistently adhering to the 2.0 architecture, the code contained within that control string standard choice is limited to debug control and a series of branch statements specific to the record type being processed during the event. All business-specific rules are contained in other standard choices beginning with the event prefix defined in the control string (in this case “ASA”). Typically the only risk associated with these projects occurs when the agency has modified the branch order in a way that does not match the 3.0 doScriptActions() execution order.

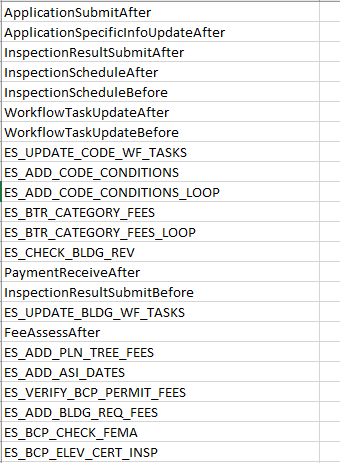


* It is not uncommon to find a stray line or two of business specific code located within the control strings. If it is limited to a small number, it can be said that the agency is still generally adhering to the 2.0 approach.

## Upgrading from 1.x

Some agencies still have not upgraded to the 2.0 architecture. Their upgrade requires a closer examination of the master scripts and careful attention to the execution order of business-specific rules. These agencies can be characterized as follows:

* No INCLUDES files. The master scripts associated to each event contain local copies of the master script functions at the bottom of the master script. Upgrading 1.x agencies requires examining these functions carefully to determine if the functions have been customized within specific scripts. It is recommended that Notepad++ Compare tool is used to locate these instances and that they be relocated to INCLUDES\_CUSTOM and named distinctively. This will require that all calls made to that function during the duration of the event be updated with the new custom function name.
* No variable branching. Agencies still adhering to the 1.x methodology have no standard choices that begin with a prefix. The business specific rules are found within the control string standard choice called by the master script and will often have “CMN” or “ES\_” branches called for iterative or repetitive code invoked by numerous branches. 1.x upgrades require that careful consideration be given to ensure that the process of breaking up the rules into branches does not adversely impact the execution order timing of rules that may build upon one another.



## Upgrading from Hybrid approach

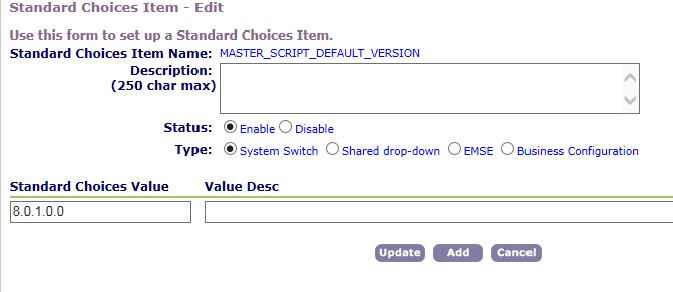
Upgrading an agency employing a hybrid approach is perhaps the most challenging as it requires breaking apart branches from a control string standard choice and ensuring consistent timing with any branches being called. It is not uncommon to find agencies that have embraced usage of the INCLUDES files but still have quite a lot of business-specific code located in the control string standard choice. These agencies can be deemed a hybrid approach and the structure of the existing code must be carefully analyzed to identify potential risks associated with segregating code into branches and execution timing.

# Existing Product Version

Depending on which product version the client is on at the time of the 3.0 upgrade, Productized 3.0 master scripts may not be available. The 3.0 Platform and associated master script versions were first deployed in version 8.0.1.0.0 and have been the preferred platform since. If the agency being upgraded is not yet using 8.0.1 or higher, the package must be deployed as a non-productized solution.

# Productized vs Non-Productized Usage

When possible, events should be associated to *productized* 3.0 Scripts (Master Script 8.0.1.0.0 or higher) during upgrade. Productized master script usage ensures that clients do not modify the Master Scripts resulting in errors. To ensure proper function, the following Standard Choice must also be deployed as shown with a value of 8.0.1.0.0 or higher matching a version of the INCLUDES\_ACCELA\_FUNCTIONS available in the product.



# Non-Event Script Types

## Batch Scripts

In most cases, it is unnecessary to perform any modification to the Batch Scripts currently employed by the agency. Many projects simply require identifying batch scripts in use or deployed within the agency and ensuring that the most recent version of those scripts is deployed to the Batch folder of the script repository.

However, if during the course of the upgrade, the client is moved from Non-productized to Productized master scripts, the following action should also be taken to ensure that the Batch Scripts continue to function should the non-productized INCLUDES\_ACCELA\_FUNCTIONS, etc… ever be removed.

1. Examine the batch scripts for instances of “INCLUDES”.
2. If a batch script is using the INCLUDES files, add the following function and replace the calls of getScriptText with getMasterScriptText.
   1. Note: INCLUDES\_BATCH, INCLUDES\_CUSTOM\_GLOBALS, and INCLUDES\_LICENSES are NOT productized and those should not be replaced with getMasterScriptText()

**function** getMasterScriptText(vScriptName){

vScriptName = vScriptName.toUpperCase();

**var** emseBiz = aa.proxyInvoker.newInstance("com.accela.aa.emse.emse.EMSEBusiness").getOutput();

**var** emseScript = emseBiz.getMasterScript(aa.getServiceProviderCode(),vScriptName);

**return** emseScript.getScriptText() + "";

}



## Set Scripts

Like the Batch Scripts, Set scripts require no conversion and typically are only identified and backed up to the repository.

As with the Batch Scripts, if the INCLUDES files are being leveraged (less common), the same getMasterScriptText() replacement should be employed.

## Pageflow Scripts

Pageflow scripts require significant attention for 3.0 upgrades, particularly when they invoke a standard choice. The code in the standard choice (if only referenced by the pageflow script) should be converted to JavaScript and inserted into the Pageflow Script being called by the trigger in ACA Admin.

For UAT and at production deployment, it is important to update the configuration with the revised script and to add the details of that configuration change to the deployment checklist.

## Custom Script vs INCLUDES\_CUSTOM

It is not uncommon to find agencies where custom functions are located both in the Custom Script (productized INCLUDES\_CUSTOM) area of Classic and a non-productized script INCLUDES\_CUSTOM. The 3.0 platform makes use of the Productized Custom Script and all custom functions deployed using the EMSE Tool will be place in that location. Non-Productized INCLUDES\_CUSTOM files should be deleted to eliminate confusion once all functions are consolidated within the single script.

# Tools

## Microsoft Access Conversion Tool Package

For agencies strictly adhering to the 2.0 EMSE Methodology (solid variable branching, few instances of business rules in the control string standard choice, consistent usage of the INCLUDES files), a Microsoft Access Conversion Tool Package has been developed to extract the existing configuration and quickly and easily build .js files.

Separate instructions are provided for that package.

## BizDomain Converter

In most cases (1.x and Hybrid upgrades) a BizDomain Converter utility script may be used to identify and convert the standard choices in use within an agency to JavaScript. This code can be placed into the Script Text window of the Script test feature and executed. The output produced will be a close (but imperfect conversion of the active standard choices and their active lines to JavaScript. It should be noted that often EMSE Standard Choices are mislabeled and a review of standard choices should be done before running this utility to make sure that all code required is converted. This code can be found within the Miscellaneous folder of the Master Script Release Distribution available on Accela Community.

// run in script test to convert std choices to scripts

// NOT PERFECT!!! please review each script

// branches will need to be converted to functions and manually updated.

var b = aa.proxyInvoker.newInstance("com.accela.aa.aamain.systemConfig.BizDomainBusiness").getOutput()

bl = b.getRBizDomains(aa.getServiceProviderCode()).toArray();

for (i in bl) { if (bl[i].getType() && bl[i].getType().equals("EMSE")) {

//aa.print(bl[i].getBizDomain().replace(":",";").replace("/","!").replace("\*","~") + "^" + "`");

aa.print("================================ " + bl[i].getBizDomain());

convert(bl[i]);

//aa.print("`" + "^");

}

}

function convert(strControl) {

var bizDomScriptResult = aa.bizDomain.getBizDomain(strControl.getBizDomain());

if (bizDomScriptResult.getSuccess())

{

bizDomScriptArray = bizDomScriptResult.getOutput().toArray()

var disableTokens = false;

var ifStatement = false;

for (var i in bizDomScriptArray)

{

if (bizDomScriptArray[i].getDescription()) {

l = bizDomScriptArray[i].getDescription().trim().split("\\^");

if (l[0].length() && ifStatement) {

ifStatement = false;

aa.print("\t}\n"); // finish continuation

}

if (l[0].length() && l[0] != "true" && l[0] != "true ") {

ifStatement = true;

if (!disableTokens) {

l[0] = l[0].replace("{","AInfo['");

l[0] = l[0].replace("}","']");

}

aa.print("if (" + l[0].trim() + ") {")

}

else {

if (l[0].length()) ifStatement = false;

}

if (l[1]) {

if (l[1].indexOf("disableTokens=true")) disableTokens = true;

if (l[1].indexOf("disableTokens=false")) disableTokens = false;

lt = l[1].trim().split(";");

for (var j in lt) {

if (!disableTokens) {

lt[j] = lt[j].replace("{","AInfo['");

lt[j] = lt[j].replace("}","']");

}

aa.print((ifStatement ? "\t" : "") + lt[j].trim() + ";");

}

}

if (ifStatement && l[2]) {

aa.print("\t} else {");

if (l[2].indexOf("disableTokens=true")) disableTokens = true;

if (l[2].indexOf("disableTokens=false")) disableTokens = false;

lt = l[2].trim().split(";");

for (var j in lt) {

if (!disableTokens) {

lt[j] = lt[j].replace("{","AInfo['");

lt[j] = lt[j].replace("}","']");

}

aa.print("\t" + lt[j].trim() + ";");

}

}

}

}

if (ifStatement) aa.print("\t}");

}

}

## Source Code Control System Repository

It is the responsibility of the agency to deploy a Source Code Control System(SCCS) that is supported for use with the EMSE Tool. Such repository shall be GIT/SVN standard and accessible through simple authentication username/password. Solutions that have proven compatible are GitHub, Assembla, CloudForge, and on-premise internal solutions such as Visual Subversion Server.

Some solutions such as Visual Studio Team Services (Microsoft) use a personal token authentication. These tokens are used to obscure an actual user’s password and provide access to the repository for limited periods of time. Settings include 180 days and 1 year. This is often a good solution for agencies that are Microsoft loyal, but requires semi-annual or annual maintenance as the EMSE Tool will cease to function when the token expires.

The current SCCS in use by the agency should be identified at the project inception and compatibility needs to be ascertained. If the solution is not compatible with the EMSE Tool, the agency should be made aware immediately to provide them time to secure a compatible solution or develop an alternative change management protocol for EMSE development and deployment.

If the agency does not have a suitable SCCS in place at the start of the project, an Assembla or GitHub repository can be created as a temporary location for converted code. The Shared Services delivery team can create a repository in Assembla for temporary usage.

## IDE – Eclipse or Notepad++ with Tortoise SVN

When converting, developing, and maintaining 3.0 scripting, it is preferable to use an Integrated Development Environment. Tools that are popular include: Visual Studio, Eclipse, or Notepad++ with Tortoise SVN plug-in.

At a minimum, it is preferable to have a tool that:

* Tools that integrate with and push/pull/refactor/merge with GIT or SVN repositories
* Formats JavaScript natively or with the use of a view or plug-in

Other useful functions include:

* Auto-fills variable names based on variables declared within the same file
* Tools that highlight syntax errors with JavaScript
* Includes Search Capabilities across multiple repositories
* Allows you to compare two files side by side

Notepad++ and Eclipse are both commonly used throughout the analysis process.

# Agency Repository and Civic Platform Integration

It should be noted that for self-hosted clients who conceal their implementations behind a firewall, adjustments need to be made to enable the av.web and av.biz servers to communicate on port 80 or 443 with an external repository. Hosted clients are enabled for this functionality natively. Some repository solutions may have additional layers of server access permissions that need to be addressed within that specific product. Consult that product’s instructions for more information.

# Process

## Data Collected from the Agency

It is recommended that the following information be collected from the agency from Production. This information may also be collected early in the project to properly assess project scope and potential risks.

* Events and their current associated master scripts
* Scripts – All
* Standard Choices – All
* Script Batch Jobs and their associated Scripts
* Scripts associated to Sets
* Script Mode Expressions
* Pageflow Scripts in Use
* Copy of content of Classic > Event > Custom Script copied to a .txt file

This can be provided via Data Manager Export for most elements but may require accessing their environment to evaluate what is in use. Self-hosted clients may also provide this data by querying the following tables and providing the information in Excel.

* Events – REVT\_AGENCY\_EVENT, Non-Productized Masters will not have a value in the UDF1 column. Productized Master Scripts will have the version number denoted in UDF1
* Scripts – REVT\_AGENCY\_SCRIPT (These may be truncated, manual retrieval from the agency may be required for Batch, Set, Pageflow, or other Customized Scripts)
* Standard Choices – RBIZDOMAIN
* Standard Choice Values – RBIZDOMAIN\_VALUE, look for all that contain a ‘^’ and make sure that these *and only these* are listed as EMSE in the RBIZDOMAIN table.
* Batch Job Configuration – BATCH\_JOB where SCRIPT\_CODE IS NOT NULL
* Set Type Configuration – XSET\_TYPE\_ATTRIBUTE, where ATTRIBUTE\_TYPE = ‘SCRIPT’, the script name will be listed in the ATTRIBUTE\_ID column.
* Expressions – REXPRESSION where EXPRESSION\_MODE = ‘Manual’ This is just to verify the presence of Manual or Script Mode expressions that may be backed up within the repository. Currently there is no direct deployment vehicle but backing up script/manual mode expression code is a good practice for process automation reference and version control. The content is a CLOB and is often truncated upon export. Retrieval from the agency may be required.
* Pageflow Configuration – RPF\_PAGE, where ONLOAD\_SCRIPT\_NAME, BEFORECLICK\_SCRIPT\_NAME, AFTERCLICK\_SCRIPT\_NAME fields are not null.

## Identifying Conversion Risks

Configuration or EMSE Changes

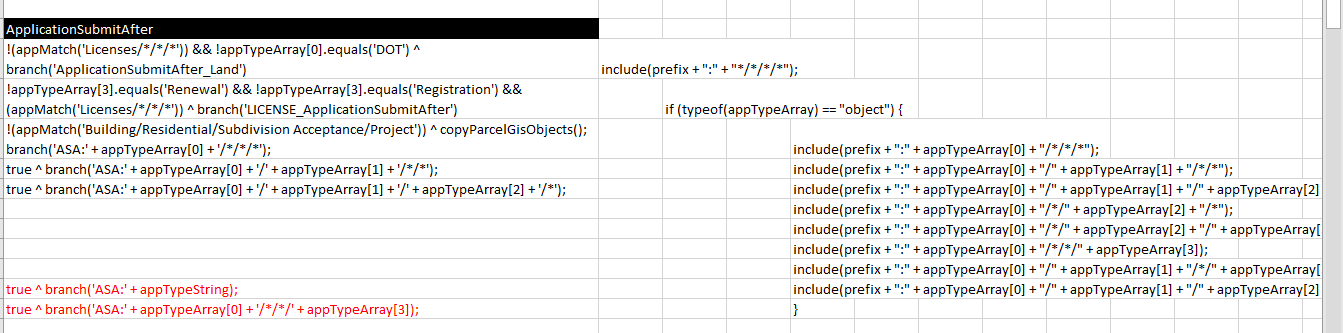
It is not uncommon for agencies to require immediate EMSE or configuration corrections during the course of an upgrade project. It is recommended that any non-critical fixes be deferred, however, if that is not possible, the agency should notify the Accela resource performing the upgrade of any and all changes required.

Branch Order Execution

Unless the agency is strictly adhering to best practice 2.0 Variable Branching and the execution order of those branches matches the doScriptActions script order, there may be a risk of shuffling the execution order of specific rules when new branches are created for Variable Branching. It is sometimes the case where a rule executed in a later branch has a dependency on action taken in an earlier branch executed during the event. Maintaining this dependent sequence requires careful analysis. This risk is increased in cases where the agency has deviated from best practice order in their 2.0 Control String standard choices.

For example: executing <eventPrefix> + ‘:’ + appTypeArray[0] + “/\*/\*/\*” last in the sequence instead of first or second. This would be particularly problematic when that standard choice invoices all fees assessed in more record specific branches executing before it. The order would be reversed when converted to 3.0 and invoicing would happen before assessment.

It is critical to be on the lookout for execution order conflicts when converting.



EMSE Standard Choices Not Marked appropriately

It is not uncommon for agencies to have Standard Choices containing executable code improperly marked as Shared Dropdowns or System Switch. Most conversion tools in our toolbox contain code that looks specifically for Standard Choices marked as “EMSE”. Before running those utilities, examine the Standard Choice values and filter the data where the VALUE\_DESC contains the caret ‘^’ symbol. This will positively identify EMSE Standard Choices. Any erroneously marked as EMSE that do not contain that symbol need to be changed to either System Switches or Shared Dropdowns as appropriate. It is important to perform this cleanup before running any of the bizdomain converter tools.

## Analysis of 1.x to 3.0

For clients upgrading directly from 1.x to 3.0 the process for conversion is not quite as straightforward. Effectively it is two upgrades combined. The following process should be followed:

* Ask the agency for the following tables in Excel format
  + REVT\_AGENCY\_EVENT (event to script mappings)
  + REVT\_AGENCY\_SCRIPT (event scripts)
  + RBIZDOMAIN (standard choices)
  + RBIZDOMAIN\_VALUE (standard choice values)

Step 1: Examine the Master Script Code associated to Active Events

By examining the REVT\_AGENCY\_EVENT tab, identify the active events and Master Scripts Associated. Any that have no Master Script Associated should be disabled as they are not executing code.

The agency’s existing Master Scripts must be examined for modifications. Such customizations may be in the form of:

* Modified master script functions that need to be migrated to INCLUDES\_CUSTOM
* Hard-Coded Variables within Master Script Functions that should be moved to INCLUDES\_CUSTOM\_GLOBALS and called by variable name
* General operational discrepancies compared to current master script versions for the same event.

The code for 1.x generation Master Scripts consisted of Event execution code (today known as the current Master Script for that event), INCLUDES\_ACCELA\_GLOBALS, INCLUDES\_ACCELA\_FUNCTIONS(\_ASB), as well as customized code that is stored in INCLUDES\_CUSTOM in the 3.0 Architecture. Each section for the active Master Scripts (Events) needs to be compared to the current version to identify customizations.

\*\*Note: Today’s Master Scripts have evolved significantly since the 1.x Platform was in place. You will find new Master Script functions, new global variables, and more robust functionality within the Master Scripts and Includes files.

The goal is to not break the event or eliminate defined variables or existing custom functions or processes by converting. Assume that the Master Script Code has been modified in some manner.

Step 2: Identify the Control Strings invoked by each Master Script

The Control String Standard Choice needs to be identified in each Master Script. These contain the automations and validations that are currently being enforced by the system in production and are the starting point for identifying and converting branches.

Step 3: Identify EMSE Standard Choices (active or not) for Documentation

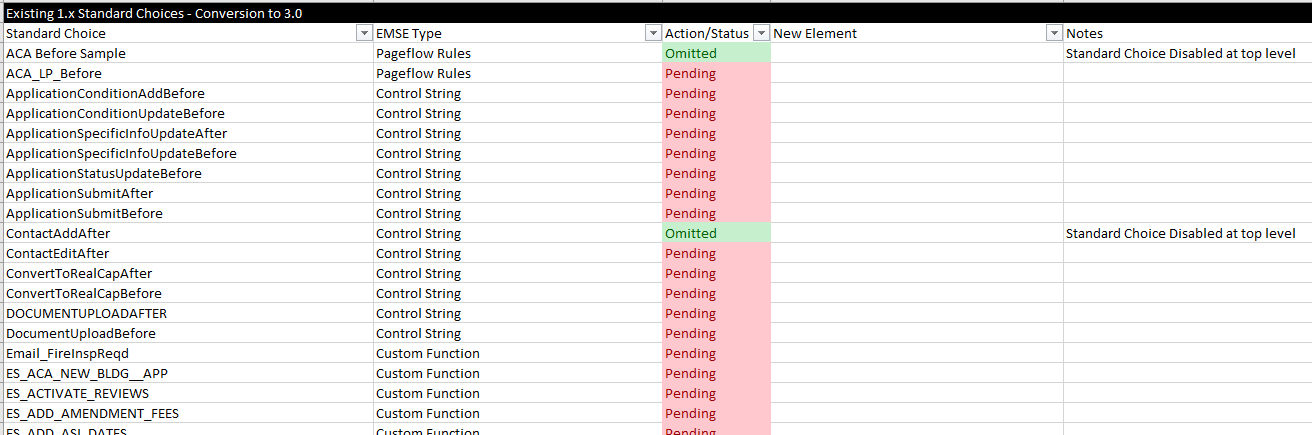
The following process should be taken to identify the Standard Choices in use by the agency and being executed by the enabled events.

* Open the RBIZDOMAIN\_VALUE worksheet
* Filter on VALUE\_DESC contains ‘^’ and SERV\_PROV\_CODE = <agencyName>
* Sort by BIZDOMAIN then BIZDOMAIN\_VALUE ascending
* Copy the first column to a new worksheet and eliminate duplicates.
* This is a list of EMSE Standard Choices within the agency, add to Conversion Details Workbook in the Existing Standard Choices section.

Step 4: Identify/Disable Inactive Branches (standard choices)

This step identifies which standard choice lines are not executing because the Standard Choice as a whole has been disabled. You will also disable any stray BIZDOMAIN\_VALUE lines for those standard choices that have been left active.

* Open the RBIZDOMAIN\_VALUE worksheet
* Filter on the agency code and REC\_STATUS = “I”
* Open the RBIZDOMAIN\_VALUE worksheet
* Disable (set to I) the RBIZDOMAIN\_VALUE entries for any disabled standard choice (REC\_STATUS is I in RBIZDOMAIN sheet). Note these in the release notes/Conversion Details Documentation.
* Label the EMSE Type of remaining Standard Choices appropriately



Step 5: Create a Conversion Directory (File Structure)

Create a 3.0 Scripting Best practices folder structure locally within your workspace.

**The following steps should be taken if you receive Data Manager Files of the Standard Choices of the Client’s environment:**

Step 6a: Import Standard Choices into a Sandbox environment

Import the client’s standard choices into a clean environment (no existing standard choices).

***\*\*NOTE: Omit this step if you have direct access to client environment\*\****

Step 7a: Clean Standard Choices

Examine standard choices to ensure that any that contain executable code are correctly marked as EMSE and those that do not are marked either System Switch or Shared Dropdown as appropriate.

Step 8a: Run BIZDOMAIN Converter in Script Test

Execute the following using Script Test and save the output.

// run in script test to convert std choices to scripts

// NOT PERFECT!!! please review each script

// branches will need to be converted to functions and manually updated.

var b = aa.proxyInvoker.newInstance("com.accela.aa.aamain.systemConfig.BizDomainBusiness").getOutput()

bl = b.getRBizDomains(aa.getServiceProviderCode()).toArray();

for (i in bl) { if (bl[i].getType() && bl[i].getType().equals("EMSE")) {

aa.print(bl[i].getBizDomain().replace(":",";").replace("/","!").replace("\*","~") + "^" + "`");

convert(bl[i]);

aa.print("`" + "^");

}

}

function convert(strControl) {

var bizDomScriptResult = aa.bizDomain.getBizDomain(strControl.getBizDomain());

if (bizDomScriptResult.getSuccess())

{

bizDomScriptArray = bizDomScriptResult.getOutput().toArray()

var disableTokens = false;

var ifStatement = false;

for (var i in bizDomScriptArray)

{

if (bizDomScriptArray[i].getDescription()) {

l = bizDomScriptArray[i].getDescription().trim().split("\\^");

if (l[0].length() && ifStatement) {

ifStatement = false;

aa.print("\t}\n"); // finish continuation

}

if (l[0].length() && l[0] != "true" && l[0] != "true ") {

ifStatement = true;

if (!disableTokens) {

l[0] = l[0].replace("{","AInfo['");

l[0] = l[0].replace("}","']");

}

aa.print("if (" + l[0].trim() + ") {")

}

else {

if (l[0].length()) ifStatement = false;

}

if (l[1]) {

if (l[1].indexOf("disableTokens=true")) disableTokens = true;

if (l[1].indexOf("disableTokens=false")) disableTokens = false;

lt = l[1].trim().split(";");

for (var j in lt) {

if (!disableTokens) {

lt[j] = lt[j].replace("{","AInfo['");

lt[j] = lt[j].replace("}","']");

}

aa.print((ifStatement ? "\t" : "") + lt[j].trim() + ";");

}

}

if (ifStatement && l[2]) {

aa.print("\t} else {");

if (l[2].indexOf("disableTokens=true")) disableTokens = true;

if (l[2].indexOf("disableTokens=false")) disableTokens = false;

lt = l[2].trim().split(";");

for (var j in lt) {

if (!disableTokens) {

lt[j] = lt[j].replace("{","AInfo['");

lt[j] = lt[j].replace("}","']");

}

aa.print("\t" + lt[j].trim() + ";");

}

}

}

}

if (ifStatement) aa.print("\t}");

}

}

**Skip to** [Step 10](#step10)

**The following Steps should be taken if you do not have direct access to the client’s environment (DB table extracts only)**

Step 6b: Prepare the worksheet for JavaScript format conversion

Complete the following steps to prepare the RBIZDOMAIN\_VALUE spreadsheet contents for conversion:

* Use find/replace to replace all double quotes with single quotes
* For the BIZDOMAIN\_VALUE column, use the custom text format of \”A@\” or, rename the column and create a new BIZDOMAIN\_VALUE column that has a formula of =CONCATENATE("A",TEXT(C5874,"000"))

***NOTE: We need to get this column to convert properly to the right sort order. This may take some experimentation, make sure you check the result in the next step. If this column is converted as an integer you will have problems.***

* Copy the filtered data to the clipboard, then use the following page: <http://www.csvjson.com/csv2json> to convert it to JSON

Step 7b: Convert JSON file to standard choices

Complete the following steps to convert the JSON object to Standard Choices

* Log into a Sandbox Environment (clean)
* Paste the JSON array into the definition of variable ‘x’ in the following code.
* Run the following in Script Test in the Clean Sandbox environment with the “Use User Transaction” value selected.

***\*\*Note, need to modify this code to set the EMSE flag for newly created standard choices\*\****

**var** x = [

{

"BIZDOMAIN": "JOHNTEST2",

"STD\_CHOICE\_TYPE": "SystemSwitch",

"BIZDOMAIN\_VALUE": "Value 1",

"REC\_STATUS": "A",

"VALUE\_DESC": "Value 1 Description!"

},

{

"BIZDOMAIN": "JOHNTEST2",

"STD\_CHOICE\_TYPE": "SystemSwitch",

"BIZDOMAIN\_VALUE": "Value 2",

"REC\_STATUS": "A",

"VALUE\_DESC": "Value 2 Description!"

}

]

**var** args = **new** Array();

**var** bm = aa.proxyInvoker.newInstance("com.accela.aa.aamain.systemConfig.RBizDomainModel", args).getOutput();

**var** r = aa.proxyInvoker.newInstance("com.accela.aa.aamain.systemConfig.BizDomainBusiness").getOutput();

**for** (**var** i **in** x) {

row = x[i];

bm.setServiceProviderCode(aa.getServiceProviderCode());

bm.setBizDomain(row.BIZDOMAIN);

bm.setDescription("");

bm.setType(row.STD\_CHOICE\_TYPE);

bm.setAuditDate(**new** Date());

bm.setAuditID("ADMIN");

bm.setAuditStatus(row.REC\_STATUS);

**try** {

r.createRBizDomain(bm);

} **catch** (err) {

aa.print("error creating biz: " + row.BIZDOMAIN + ":" + row.BIZDOMAIN\_VALUE)

}

z = aa.bizDomain.createBizDomain(row.BIZDOMAIN, row.BIZDOMAIN\_VALUE, row.REC\_STATUS, row.VALUE\_DESC);

aa.print(row.BIZDOMAIN + ":" + row.BIZDOMAIN\_VALUE + " = "+ z.getSuccess());

}

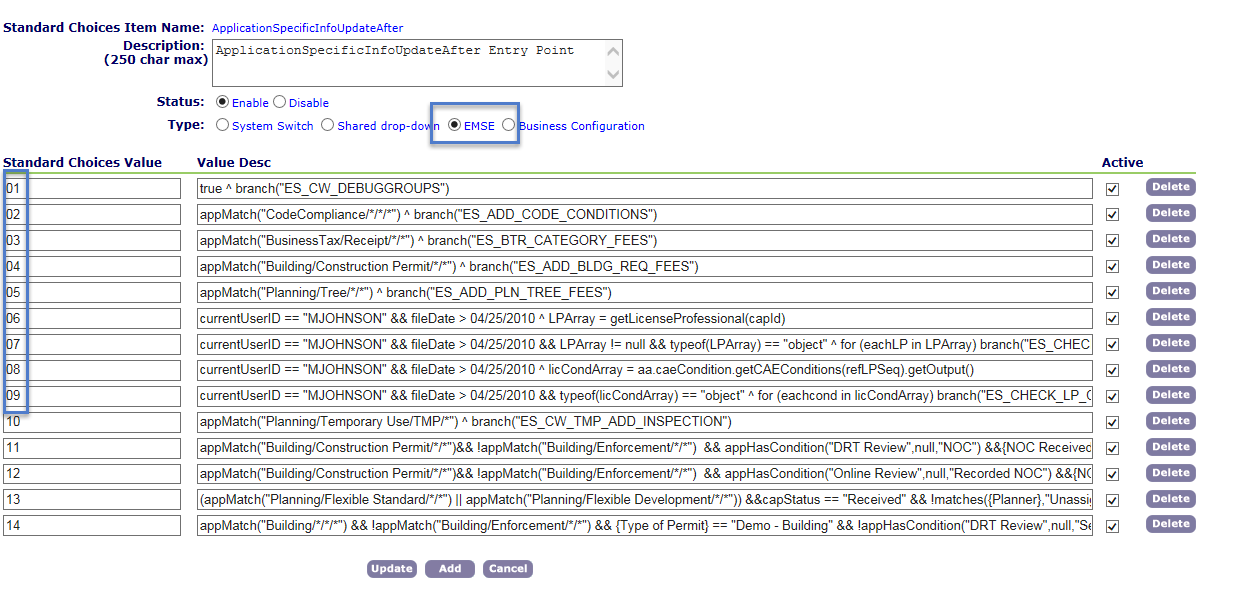
Step 8b: Examine and set EMSE flag for the new standard choices

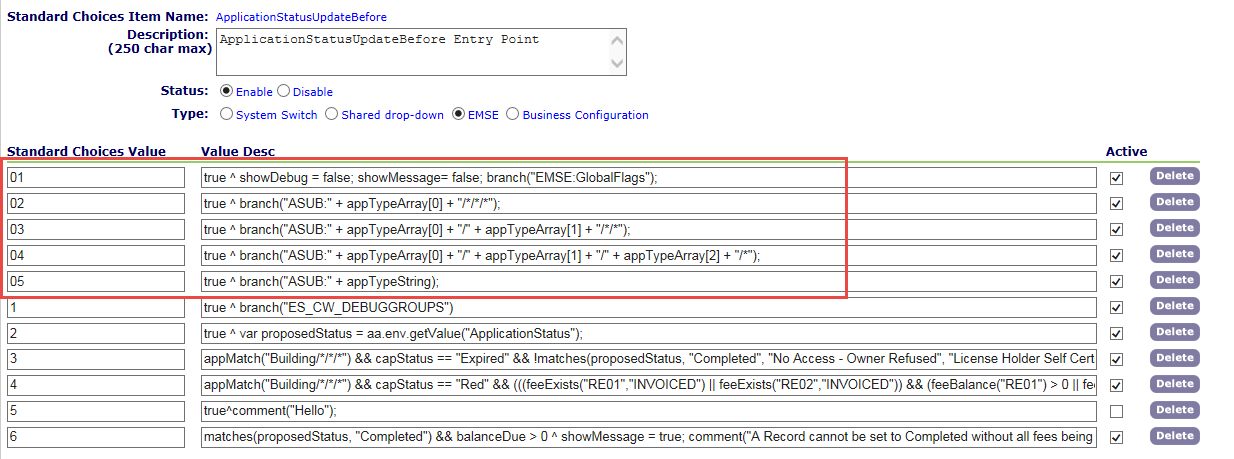
Disabled lines will not be created. It is important that the Standard Choice values be examined for sort order (you may need to insert leading zeros) and ensure that the EMSE flag is set.

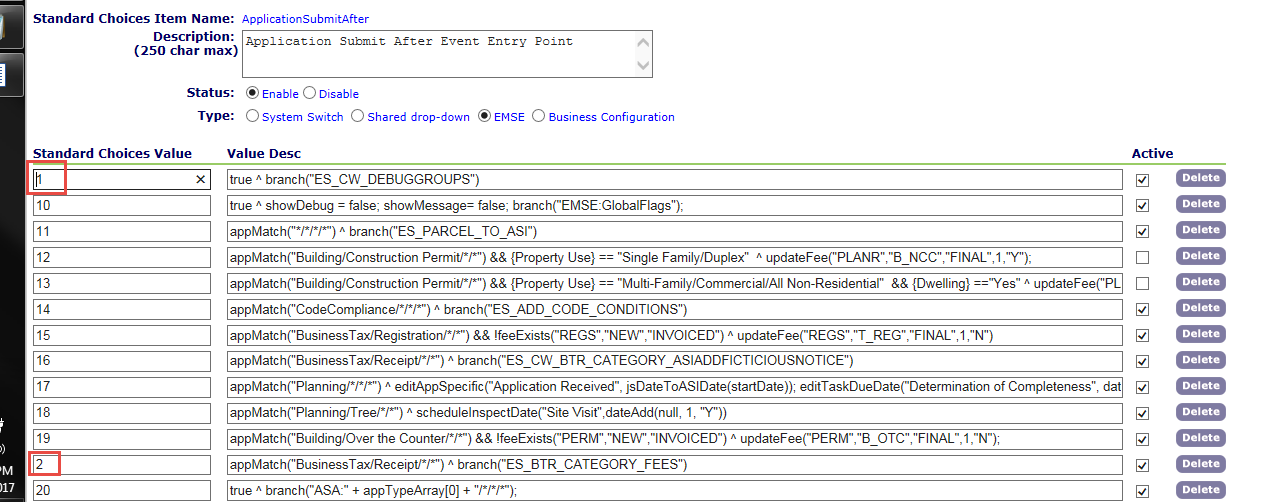
If the environment was not “clean” (BPT standard choices pre-exist), compare each EMSE standard choice with the source data to make sure that BPT rows are eliminated. Pay particular attention to:

* lines 10, 20, 30, etc… of Application Submit After, the existing values will not be overwritten.
* Case differentiation within the Standard Choice name will also prevent the values to not successfully create, for example WORKFLOWTASKUPDATEAFTER and WorkflowTaskUpdateAfter.

***\*\*Note: this step could be omitted if the loader were improved\*\****







Step 9b: Run BIZDOMAIN Converter in Script Test

Execute the following using Script Test and save the output.

// run in script test to convert std choices to scripts

// NOT PERFECT!!! please review each script

// branches will need to be converted to functions and manually updated.

var b = aa.proxyInvoker.newInstance("com.accela.aa.aamain.systemConfig.BizDomainBusiness").getOutput()

bl = b.getRBizDomains(aa.getServiceProviderCode()).toArray();

for (i in bl) { if (bl[i].getType() && bl[i].getType().equals("EMSE")) {

aa.print(bl[i].getBizDomain().replace(":",";").replace("/","!").replace("\*","~") + "^" + "`");

convert(bl[i]);

aa.print("`" + "^");

}

}

function convert(strControl) {

var bizDomScriptResult = aa.bizDomain.getBizDomain(strControl.getBizDomain());

if (bizDomScriptResult.getSuccess())

{

bizDomScriptArray = bizDomScriptResult.getOutput().toArray()

var disableTokens = false;

var ifStatement = false;

for (var i in bizDomScriptArray)

{

if (bizDomScriptArray[i].getDescription()) {

l = bizDomScriptArray[i].getDescription().trim().split("\\^");

if (l[0].length() && ifStatement) {

ifStatement = false;

aa.print("\t}\n"); // finish continuation

}

if (l[0].length() && l[0] != "true" && l[0] != "true ") {

ifStatement = true;

if (!disableTokens) {

l[0] = l[0].replace("{","AInfo['");

l[0] = l[0].replace("}","']");

}

aa.print("if (" + l[0].trim() + ") {")

}

else {

if (l[0].length()) ifStatement = false;

}

if (l[1]) {

if (l[1].indexOf("disableTokens=true")) disableTokens = true;

if (l[1].indexOf("disableTokens=false")) disableTokens = false;

lt = l[1].trim().split(";");

for (var j in lt) {

if (!disableTokens) {

lt[j] = lt[j].replace("{","AInfo['");

lt[j] = lt[j].replace("}","']");

}

aa.print((ifStatement ? "\t" : "") + lt[j].trim() + ";");

}

}

if (ifStatement && l[2]) {

aa.print("\t} else {");

if (l[2].indexOf("disableTokens=true")) disableTokens = true;

if (l[2].indexOf("disableTokens=false")) disableTokens = false;

lt = l[2].trim().split(";");

for (var j in lt) {

if (!disableTokens) {

lt[j] = lt[j].replace("{","AInfo['");

lt[j] = lt[j].replace("}","']");

}

aa.print("\t" + lt[j].trim() + ";");

}

}

}

}

if (ifStatement) aa.print("\t}");

}

}

**\*\*Steps from this point on are consistent regardless of data source for 1.x upgrades\*\***

Step 10: Identify frequency of branch calls

For any non-control string standard choices, search the RBIZDOMAIN\_VALUE, VALUE\_DESC column for incidences of call. If the branch is only called once, insert the content into the branch calling it. This is common for BRANCH\_NAME, BRANCH\_NAME\_LOOP combinations as the 1.x and 2.0 architecture were not supportive of multiline nested logic within for loops. If the branch is called more than once, it should be noted to convert it as a custom function to eliminate redundancy. If the branch is not called, indicate it as omitted and list in the Omissions section of the release notes.

Step 11: Omit “Debug only” Standard Choices

You may find that the only active lines within a standard choice are lines to turn debug or messaging on or off. As debug control has been centralized in INCLUDES\_CUSTOM\_GLOBALS and set by the user, such debug control is poor practice. Do not convert these standard choices and note them in the release notes.

## Format and Best Practices

When converting the standard choices, 3.0 EMSE Best Practices should be kept in mind.

Debug Control

All instances of showDebug need to be removed from the new or converted branches. The 3.0 architecture supports and encourages global debug settings ***by user***. The INCLUDES\_CUSTOM\_GLOBALS script should be deployed and administrative users should be added so that debug is on for administrators by default and off for daily users.

Comments

Comments should be included where appropriate to indicate where branches have been added inline to code when only called by a single script and to define the conversion resource assigned the script when multiple consultants make up the project team.

JavaScript Layout and Indentation

Accela standards for indentation and spacing should be observed to deliver an easy to read, clean solution to the client. While compressing files may result in faster execution, the client is often left unable to maintain and troubleshoot the code moving forward.

Elimination of Redundant Code/Custom Functions

Where appropriate, branches called from multiple scripts are to be converted to Custom Functions. If only called in one place, it is to be placed inline. Be on the lookout for redundant code. While it isn’t in project scope to significantly modify or make changes to the scripts outside of supporting variable branching or 3.0 architectural standards, any noted redundancy should be included in the Suggestions portion of the Analysis document.

Try/Catch

New scripts and custom functions should be wrapped in try/catch blocks with appropriate debug messaging indicating the script and line number(or stack trace) of failure.

# Documentation

A release notes document summarizing the conversion analysis shall be provided. This document should include the following elements

## Items Omitted from Conversion

Disabled Standard Choices or Branches not called by active Standard choices should not be added to the repository. These may be included as a supplemental file in the final deliverable for reference. These typically include abandoned development approaches or code to enforce business practices that are no longer relevant in the agency’s operations. Other entries in this section include the Control String Standard Choices for 2.0 upgrades where no deviation from execution order was observed and no business specific code was present.

## Suggestions for Cleaning Up Code

Where apparent, the consultant should include a section in the analysis document that describes areas where improvement can be made to:

* Eliminate redundancy of code – function creation with the passing of variables
* Prevent unnecessary heap memory usage (such as building the elements of an email message before the code decides whether it will be sent)
* Eliminate hard-coding of values such as emails or staff assignments. (Lookup function and/or INCLUDES\_CUSTOM\_GLOBAL variables).
* Etc….

## Existing Errors in Code

Correcting existing errors in code execution or logical holes are outside of project scope. However, if an error is discovered during UAT that is present in the control environment (2.0), the issues should be noted, and if the resolution is simple, it should be detailed in a section of the analysis document.

## Unused Events

It is not uncommon to find unused events deployed within an agency. If there are no specific rules created for a particular event, the event may be disabled.

## Unit Test Notes and Observations

Before releasing to the client for UAT it is best to perform unit tests of all converted events to ensure proper execution.

# User Acceptance Testing

User Acceptance Testing shall be performed by the agency and may include the following:

* Working through existing test scripts that the agency uses for upgrades. Critical workflow processes should be exercised by Subject Matter Experts.
* Examining the Script Inventory/Repository and conducting specific positive/negative/boundary value tests based on rules evident in the code.
* Use of Script Test Feature to simulate events under various task/status, inspection/result, application submit scenarios.