Accela Automation® Event Manager Script Engine Master Scripts version 1.6
FUNCTION LIST

Note that some functions are only available in master scripts where they are applicable.
All function names and parameter values are case sensitive, unless otherwise stated.
New functions introduced in 1.6 are shaded

Function	Ver	Category	Type	Parameters	Description
activateTask (wfTask, [wfRelationSeqId])	1.3	workflow	edit	*wfTask (string)* Name of task to be made active. *wfRelationSeqId (number: long)(optional)* Relation sequence ID of workflow process that *wfTask* belongs to.	Makes workflow task *wfTask* active and not completed, so that *wfTask* can be edited by users. If workflow uses sub-processes that contain duplicate workflow task names, use parameter ***wfRelationSeqId*** to specify the process or subprocess whose wfTask should be activated. The value of ***wfRelationSeqId*** can only be found by querying workflow tables (e.g. GPROCESS.RELATION_SEQ_ID)
addAddressCondition (addNum, cType, cStatus, cDesc, cComment, cImpact)	1.6	Condition	Add	*addNum (long)* *reference* address number or null *cType* Type of condition (from admin->condition->condition type) *cStatus* Status (from admin->condition->condition status) *cDesc* Description of the condition (free text) *cComment* Condition comment (free text) *cImpact:* must be “Lock”, “Hold”, “Notice”, “Required”, or “”	Adds a condition to the specificed reference address. If addNum is null, then the condition will be added to all reference addresses associated with the currnet CAP.
addAllFees (fsched, fperiod, fqty, finvoice)	1.3	Fee	Add	*fsched (string)* Fee schedule to be added *fperiod (string)* Fee period to be used *fqty (integer)* Quantity to be entered *finvoice (string)* Flag for invoicing (“Y” or “N”)	Adds all fees within a fee schedule to the application. Optionally flags the fees for automatic invoicing by the script.

Function	Ver	Category	Type	Parameters	Description
addAppCondition (cType, cStatus, cDesc, cComment, cImpact)	1.3	Condition	Add	*cType* Type of condition (from admin->condition->condition type) *cStatus* Status (from admin->condition->condition status) *cDesc* Description of the condition (free text) *cComment* Condition comment (free text) *cImpact:* must be “Lock”, “Hold”, “Notice”, “Required”, or “”	Adds the condition to the application.

addASITable
(tableName,
tableValueArray)
	1.6	ASI	Add	*tableName (string)*
				Name of the ASI table to add to the CAP
				tableValueArray (array of associative arrays)
				values to populate the table
					Populates the ASI table with values. tableValueArray is an array of arrays. Each array object within tableValueArray must contain an associative index for each column in the target table. For example:
					
					masterArray = new Array();
					elementArray = new Array();
					elementArray[“Table Column 1”] = “Row 1, column 1 Value”;
					elementArray[“Table Column 2”] = “Row 1, column 2 Value”;
					masterArray.push(elementArray);
					addASITable(“table name”,masterArray);
					
					this example will populate the 2-column table with one row.

addFee (fcode, fsched, fperiod, fqty, finvoice)	1.5	Fee	Add	*fcode (string)* Fee code to be added. *fsched (string)* Fee schedule of the fee to be added *fperiod (string)* Fee period to be used. *fqty (integer)* Quantity to be entered. *finvoice (string)* Flag for invoicing (“Y” or “N”).	Adds a single fee *fcode* to the application, from the fee schedule *fsched* with fee period *fperiod* and quantity of *fqty*. If finvoice is “Y”, the fee is invoiced. If *finvoice* is “N”, the fee is assessed but not invoiced. Returns the Fee Sequence number of the fee added. The fee period *fperiod* must be a valid fee period for *fcode* in *fsched*, else this function will throw an error. *See also*: addAllFees function.

Function	Ver	Category	Type	Parameters	Description
addFee (fcode, fsched, fperiod, fqty, finvoice, [capId])	1.3	Fee	Add	*fcode (string)* Fee code to be added. *fsched (string)* Fee schedule of the fee to be added *fperiod (string)* Fee period to be used. *fqty (integer)* Quantity to be entered. *finvoice (string)* Flag for invoicing (“Y” or “N”). *capId (CapIDModel) (optional)* CapID object.	Adds a single fee *fcode* to the application, from the fee schedule *fsched* with fee period *fperiod* and quantity of *fqty*. If finvoice is “Y”, the fee is invoiced. If *finvoice* is “N”, the fee is assessed but not invoiced. If *capId* optional parameter is used, updates application *capId*. If *capId* parameter is not used, updates current application. The fee period *fperiod* must be a valid fee period for *fcode* in *fsched*, else this function will throw an error. Hint: getApplication( ), getParent( ), createChild() functions each returns a capId object that can be used in the *capId* parameter. See Also: addAllFees function.
addFeeWithExtraData (fcode, fsched, fperiod, fqty, finvoice, feeCap, feeComment, UDF1, UDF2)	1.6	Fee	Add	*fcode (string)* Fee code to be added. *fsched (string)* Fee schedule of the fee to be added *fperiod (string)* Fee period to be used. *fqty (integer)* Quantity to be entered. *finvoice (string)* Flag for invoicing (“Y” or “N”). *feeCap (CapIDModel)* CapID object. feeComment (string) comment field on the fee item UDF1 (string) Value for user defined field on fee item UDF2 (string) Value for user defined field on fee item	Identical to the addFee function, but also allows for the comment and user defined fields to be populated.

Function	Ver	Category	Type	Parameters	Description
addLicenseCondition (cType, cStatus, cDesc, cComment, cImpact, [stateLicNum])	1.4	Professional	Add	*cType (string)* Condition type. *cStatus (string)* Condition status. *cDesc (string)* Condition (30 characters maximum) *cComment (string)* Condition comment (free text) *cImpact:* Condition severity: "Lock", "Hold", "Notice", "Required", or "". *stateLicNum (string) (optional)* State license number.	Adds the condition (*cType, cStatus, cDesc, cComment, cImpact*) to the reference record for each licensed professional on the application. If *stateLicNum* parameter is used, the function adds the condition to the licensed professional reference record whose State License Number is *stateLicNum*. This licensed professional does not have to be on the current application.
addLookup (itemName, valueName, valueDesc)	1.3	utility	Add	*itemName (string)* Standard Choices Item Name *valueName (string)* Standard Choices Value *valueDesc (string)* Standard Choices Value Description	Adds a lookup entry to an existing Standard Choices Item. Adds a new value called *valueName* with the value description of *valueDesc* to Standard Choices Item Name *itemName.* Note: If the Standard Choices Item *itemName* already has a value entry called *valueName*, *valueName* will not be added or updated. The Standard Choices Item *itemName* must already be created—this function will not create Standard Choices Item *itemName* if it does not exist.
addParcelAndOwnerFrom RefAddress (refAddress, Optional capId)	1.6	Record	Add	*refAddress (long)* *Reference address number to copy data from* *capId (capId optional)* *target CAP for parcel and owner*	Copies the associated parcel and owner from a reference address to the specified CAP. If no CAP is specified, the target will be the current CAP.
addParcelCondition (parcelNum, cType, cStatus, cDesc, cComment, cImpact)	1.3	parcel	Add	*parcelNum (string)* Parcel number that condition is added to. If **null** is used, condition will be added to all parcels on the application. *cType (string)* Condition type. *cStatus (string)* Condition status. *cDesc (string)* Condition name. *cComment (string)* Condition comment. *cImpact (string)* Condition severity.	Adds a condition to the reference parcel whose number is *parcelNum*. The condition's Type, Condition (description), Status, Severity and Comment will be *cType, cDesc, cStatus, cImpact*, and *cComment* respectively. The condition's Apply and Effective dates will be the current date. The condition's Applied By and Action By staff names will be the current user's name. If **null** is used for *parcelNum* parameter, the condition will be added to all parcels on the current application.
addParcelDistrict (parcelNum, districtValue)	1.6	Record	Add	*parcelNum (String)* *Parcel number that district is added to.* *districtValue (string)* *Value of district entry to add*	Adds a district to the parcel on a CAP. Does not edit reference parcel data. If parcelNum is null, the district will be added to all parcels on the current CAP.

Function	Ver	Category	Type	Parameters	Description
addParent (applicationNumber)	1.3	hierarchy/ related CAPs	Add	*applicationNumber (string)* App number (B1_ALT_ID) of the application to be made parent of the current application.	Adds the current application as a hierarchal child to the parent application *applicationNumber*.
addrAddCondition (addrNum, cType, cStatus, cDesc, cComment, cImpact, allowDuplicate)	1.4	Address	Add	*addrNum (number)* Address number. Use **null** for all addresses on CAP. *cType (string)* Condition type. *cStatus (string)* Condition status. *cDesc (string)* Condition name. *cComment (string)* Condition comment. *cImpact (string)* Condition severity. *allowDuplicate (string)(optional)* "N" to prevent duplicate condition added to address.	Adds a condition (*cType, cStatus, cDesc, cComment, cImpact*) to the address on the CAP whose address number is *addrNum*. If *addrNum* is **null**, adds the condition to all the addresses on the CAP. If allowDuplicate is "N", will not add a condition to the address if the same condition is already on the address. If allowDuplicate is "Y", will add the condition to the address even if the condition will be duplicated on the address as a result. Returns **true** if condition is added, **false** if no condition is added. Note: The condition is added to the reference Address record. The condition is added only if the address was added to the CAP using the *Search* button on the CAP's Address screen (AA 6.2.1 and above) or using the *Get Associated Object* button on the CAP's parcel screen. If the address was entered manually, the condition will not be added to it. Hint: The *addrNum* value comes from B3ADDRES. L1_ADDRESS_NBR, not B3ADDRES.B1_ADDRESS_NBR.
addReferenceContactByName (vFirst, vMiddle, vLast)	1.6	Contact	Add	*vFirst (string)* *first name of reference contact* *vMiddle (string)* *middle name of reference contact* *vLast (string)* *last name of reference contact*	Adds a reference contact to the current CAP, based on the name of the contact. Will only add the first matching contact.
addressExistsOnCap (capId) optional	1.6	Record	True/False	*capId (optional)* *capId to check*	Returns true if there is at least one address on the CAP
addStdCondition (cType, cDesc)	1.4	Condition	Add	*cType (string)* Condition type. *cDesc (string)* Condition name	Retrieves all standard conditions named *cDesc* whose type is *cType* and adds them to the CAP. Condition is assigned the following values: Status = Applied Applied By = current user Action By = current user Apply Date = current date Effective Date = current date Expiration Date = *blank* **Note:** Function can only be used with AA 6.4 and above.

Function	Ver	Category	Type	Parameters	Description
addToASITable (tableName, rowValues, [capId])	1.4	ASI	Add	*tableName (string)* Application specific info table name. *rowValues (array of strings)* Values for a single table row, as an associative array of strings. *capId (CapIDModel)(optional)* CapID object for application.	Adds one row of values (*rowValues*) to the application specific info (ASI) table called *tableName*. The *rowValues* parameter must be an associative array of string values, where each element name is a column name in the ASI table *tableName*, and the element stores the column value. If the *capId* parameter is used, the function adds *rowValues* to *tableName* in the CAP whose capId object is *capId*. The parameter *rowValues* does not have to contain all the columns in the ASI table *tableName*. The ASI table *tableName* must already exist on the CAP.

allTasksComplete
(wfProcess,
[igTask1,
…
igTaskn])
	1.3	workflow	true/
			false
				wfProcess (string)
				Process name of workflow to check.
				igTask1 … igTaskn (string) (optional)
				Names of tasks to ignore. Enter one or more task name parameters. Case sensitive.
					Returns **true** if all tasks (excluding tasks in optional *igTask1… igTaskn* list) in workflow process / sub-process *wfProcess* are completed. Returns **false** if any task (excluding tasks in optional *igTask1… igTaskn* list) in workflow process / sub-process *wfProcess* is not completed.
					
					Hint: *wfProcess* is R1_PROCESS_CODE in the GPROCESS and SPROCESS tables.
					
					Examples
					To determine if all tasks in workflow “BLDG” are completed:
					`allTasksComplete("BLDG")`
					
					To determine if all tasks in workflow “BLDG” are completed, ignoring “Optional Review” and “Closure” tasks:
					`allTasksComplete("BLDG","Optional Review", "Closure")`

appHasCondition (cType, cStatus, cDesc, cImpact)	1.4	Condition	true/ false	*cType (string)* Condition type. *cStatus (string)* Condition status. *cDesc (string)* Condition name. *cImpact (string)* Condition severity.	Returns **true** if the application has an application condition whose type is *cType*, name is *cDesc*, status is *cStatus*, and severity is *cImpact*; otherwise, returns **false**. Use **null** in place of any parameter if you do not want to filter by that item. E.g., To check if the application has any condition at all, use `appHasCondition(null, null, null, null)`

Function	Ver	Category	Type	Parameters	Description
appMatch (appType)	1.3	Record	true/ false	*appType (string)* Four level application type. Must contain 3 slash (/) characters. Case sensitive. Do not add spaces before or after slashes. The asterisk (*) may be used as a wildcard to match all entries for a given level.	Returns **true** if *appType* matches the current application's application type, **false** if it does not. Compares the current application type to *appType*. The asterisk (*) is used as a wildcard and can be used to match all entries for a given level. For example: appMatch("Building/*/Sign/*/*") will evaluate to True for application type "Building/Commercial/Sign/Billboard" as well as "Building/Residential/Sign/Garage Sale". Note: *appType* must contain 3 slash characters (/). Do not add spaces immediate before or after the slash (/).
appNameIsUnique (appTypeLevel1, appTypeLevel2, capName)	1.4	Record	true/ false	*appTypeLevel1 (string)* Application group (1st level of application type). *appTypeLevel2 (string)* Application type (2nd level of application type). *capName (string)* Application name to test.	Returns **true** if the application name *capName* has not been used in any other applications whose app type begins with *appTypeLevel1* / *appTypeLevel2*. Returns **false** if *capName* is not unique.
assignCap (userID, [capId])	1.5	Record	edit	*userID (string)* User ID of that CAP is to be assigned to. *capId (CapIDModel) (optional)* CapID object for CAP being assigned.	Assigns the staff whose user ID is *userID* to the current CAP. Also assigns the user's department. If optional parameter *capId* is used, assigns the staff and department to the CAP *capId* instead.
assignInspection (inspNum, userID)	1.4	Inspection	edit	*inspNum (number)* Inspection sequence number. *userID (string)* Inspector's user ID.	Assigns the inspector whose user ID is *userID* to the inspection whose sequence number is *inspNum*. The inspection must already be scheduled on the CAP.
assignTask (wfTask, userId, [wfProcess])	1.3	workflow	edit	*wfTask (string)* Workflow task to be edited. *userId (string)* UserId of staff to be assigned to *wfTask*. Case sensitive. *wfProcess (string) (optional)* Process name of workflow for *wfTask*. Case sensitive.	Assigns the staff whose user ID is *userId* to workflow task *wfTask*. No workflow history record is created. If application's workflow contains duplicate *wfTask* tasks, use parameter *wfProcess* to specify the process or subprocess whose *wfTask* should be activated. Hint: *wfProcess* is R1_PROCESS_CODE in the GPROCESS and SPROCESS tables. *UserId* and *wfProcess* are normally in uppercase.
autoAssignInspection (inspectionNumber)	1.6	Inspection	Edit	*inspectionNumber (long)* *sequence number for the inspection to assign*	Uses the system automatic inspection assignment to assign the specified inspection.
branch (choiceName)	1.3	utility	utility	*choiceName (string)* Standard Choices Item Name (string). Case sensitive.	Executes the standard choice script control whose name is *choiceName* as a sub-control. The script *choiceName* must contain only valid criteria/action pairs sequentially numbered. Example `branch("Inspection:Update Expiration")`

Function	Ver	Category	Type	Parameters	Description
branchTask (wfTask, wfStatus, wfComment, wfNote, [wfProcess])	1.3	workflow	Edit	*wfTask (string)* workflow task name *wfStatus (string)* status *wfComment (string)* comment *wfNote (string)* note to add to the workflow task *wfProcess (optional) (string)* ID (R1_PROCESS_CODE) for the process that the task belongs to. Required for multi-level workflows.	Updates the workflow task *wfTask* as follows Status = *wfStatus* Status Date = current date Status Comment = *wfComment* Action By = current user Task *wfTask* is closed and the workflow proceeds to the branch task. If application's workflow contains duplicate *wfTask* tasks, use parameter *wfProcess* to specify the process or subprocess whose *wfTask* should be edited.
capHasExpiredLicProf (dateType, [licType, capId])	1.4	Professional	true/ false	*dateType (string)* Expiration date to check. Options (use one): EXPIRE, INSURANCE, BUSINESS. *licType (string)(optional)* License type. *capId (CapIDModel)(optional)* CapID object of CAP. If not used or **null**, current application is used.	Returns **true** if any licensed professional on the CAP has expired; otherwise, returns **false**. Checks for expiration by retrieving the licensed professional reference record having the same license # and checking the expiration date specified by *dateType*. If the expiration date is on or before the current date, the script returns **true**. Skips disabled licensed professionals. Use parameter *licType* to check a specific license type. Use parameter *capId* to check licensed professionals on a CAP other than the current CAP. *dateType* — Expiration Date Field Checked EXPIRE — License Expiration Date INSURANCE — Insurance Expiration Date BUSINESS — Business License Expiration Date
capIdsFilterByFileDate (capIdArray, startDate, endDate)	1.4	Record	Get	*capIdArray (array of CapIDModel objects)* Array of CapID (CapIDModel) objects to filter. *startDate (string)* Start date of file date range, in MM/DD/YYYY format. *endDate (string)* End date of file date range, in MM/DD/YYYY format.	Searches though the CAPs in *capIdArray* and returns only CAPs whose file date is between *startDate* and *endDate*, as an array of CapID (CapIDModel) objects Hint: To find the number of CAPs returned, store the return value to a variable and use the length property to find the number of CAPs in the array. E.g. `capArray = capIdsFilterByFileDate(myCapArray, "01/01/2006", "12/31/2006"); capCount = capArray.length;`

Function	Ver	Category	Type	Parameters	Description

capIdsGetByAddr
()
	1.4	Address	Get	*(none)*	Returns CAPs that have the same property address as the current CAP, as an array of capId (CapIDModel) objects. If the current CAP has no property address, returns **false**. Addresses are matched using these fields:
					- House Nbr Start
					- Street Direction
					- Street Name
					- Street Suffix
					- Zip
					
					Function can be used with all events except ApplicationSubmitBefore. The CAPs returned include the current CAP. If the current CAP has more than one property address, the function uses the first address to match.
					
					Hint: To find the number of CAPs returned, store the return value to a variable and use the length property to find the number of CAPs in the array. Example:
					`capArray = capIdsGetByAddr();`
					`logDebug("Number of CAPs: " + capArray.length);`

capIdsGetByParcel
([parcelNum])
	1.4	parcel	Get	*parcelNum (string)(optional)*
				Parcel number to search for. If this is **null** or omitted, the first parcel number on the current CAP is used.
					Returns CAPs that have the same parcel as the current CAP, as an array of capId (CapIDModel) objects. If the current CAP has no parcel, returns ***false***.
					
					The CAPs returned include the current CAP.
					
					Hint: To find the number of CAPs returned, store the return value to a variable and use the length property to find the number of CAPs in the array. E.g.
					`capArray = capIdsGetByParcel();`
					`logDebug("Number of CAPs: " + capArray.length);`

checkCapForLicensedProfessionalType (licProfType)	1.6	Record	True/False	*licProfType(string)* *Licensed Professional Type to check for*	Returns true if a licensed professional of the type exists on the current CAP.
checkInspectionResult (inspDesc, inspResult)	1.3	Inspection	true/ false	*inspDesc (string)* Inspection to check. Case sensitive. *inspResult (string)* Inspection result (or status) to look for. Case sensitive.	Returns **true** if the inspection *inspDesc* has the result of *inspResult*, or **false** if it does not. Can also be used to check if inspection *inspDesc* is scheduled (not yet resulted), by using “Scheduled” in *inspResult*.

Function	Ver	Category	Type	Parameters	Description
childGetByCapType (appType, [parentCapId], [skipChildCapId])	1.4	hierarchy/ related CAPs	Get	*appType (string)* Four level application type. Must contain 3 slash (/) characters. Do not add spaces before or after slashes. The asterisk (*) may be used as a wildcard to match all entries for a given level. parentCapId *(CapIDModel)(optional)* CapID object for parent application. Use **null** if *skipChildCapId* parameter is used. skipChildCapId *(CapIDModel)(optional)* CapID object of child application to skip.	Searches through all child CAPs and returns the CapID object for the first child CAP whose application type matches *appType*. If the *parentCapId* parameter is used, searches child CAPs of the application whose CapID object is *parentCapId*. If the *skipChildCapId* parameter is used, function will skip over any child CAP whose CapID object is *skipChildCapId*. See also: **getChildren** function. Hint: To find the sibling of the current application, use the function getParent() as the parentCapId parameter and capId as the skiChildCapId parameter. E.g. `siblingCapId = childGetByCapType("*/*/*/*", getParent(), capId)`
closeSubWorkflow (thisProcessID, wfStat)	1.6	Workflow	Edit	*thisProcessID* *ID of the process to check* *wfStat* *Status to use when closing the parent task*	A function that is useful when working with sub-processes. It will check all the tasks in the sub-process for completeness. If all tasks are complete, it will close the parent task with the specified status. For example: closeSubWorkflow(wfProcessID,"Completed");
closeTask (wfTask, wfStatus, wfComment, wfNote, [wfProcess])	1.3	workflow	Edit	*wfTask (string)* workflow task name *wfStatus (string)* status to update *wfComment (string)* comment to add *wfNote (string)* note to add to the workflow task *wfProcess (optional) (string)* ID (R1_PROCESS_CODE)for the process that the task belongs to. Required for multi-level workflows.	Updates the workflow task *wfTask* as follows: Status = *wfStatus* Status Date = current date Status Comment = *wfComment* Action By = current user Task *wfTask* is closed and the workflow proceeds to the next task, even if *wfStatus* is set up to loop or branch. If workflow needs to loop or branch, use loopTask or branchTask functions. If application's workflow contains duplicate *wfTask* tasks, use parameter *wfProcess* to specify the process or subprocess whose *wfTask* should be edited. Note: This function used to be called closeWorkflow[2].

Function	Ver	Category	Type	Parameters	Description

comment
(commentstr)
	1.3	utility	Display	*commentstr (string)*
				comment to display
					Use logMessage and logDebug functions instead.
					
					Adds the message *commentstr* to the message/debug window when the script executes. If debug is turned on (i.e., showDebug = **true**), the comment will show in the debug messages. If messages are turned on (i.e., showMessage = **true**), the comment will show in the messages. If neither is turned on, the comment will not display.
					
					This function can be used to display messages to the user, as well as
					variables to aid in debugging issues.
					
					Use this function instead of directly assigning value to **message** variable in script control.
					
					Examples
					`true ^ comment("calcValue is " + calcValue)`
					`true ^ comment("The building fees have been added automatically")`

comparePeopleGeneric (peopleModel)	1.6	Contact	Utility	*peopleModel (peopleModel)* *peopleModel object containg the criteria*	this function will be passed as a parameter to the createRefContactsFromCapContactsAndLink function. takes a single peopleModel as a parameter, and will return the sequence number of the first G6Contact result. returns null if there are no matches In order to use attributes enhancement 09ACC-05048 must be implemented
completeCAP (userId, [capId])	1.5	Record	Edit	*userId (string)* ID of user that completes the CAP. *capId (CapIDModel)(optional)* CapID object for CAP being updated.	Assigns the staff whose user ID is *userId* to the Completed by Staff field on a CAP. Also sets the Completed by Date value to the current date. If capId optional parameter is used, updates application *capId*. If *capId* parameter is not used, updates current application.
contactAddFromUser (pUserID)	1.6	Contact	Add	*pUserID (string)* *userID used as critera to search for contact*	Searches for a reference contact that matches the supplied userID, based on first, middle, and last names. If a matching contact is found, it is added to the current CAP as a cap contact.
contactSetPrimary (pContactNbr)	1.6	Contact	Edit	*pContactNbr (long)* *sequence number of the contact to make primary*	Sets the supplied contact to be the primary contact on the current CAP

Function	Ver	Category	Type	Parameters	Description
contactSetRelation (pContactNbr, pRelation)	1.6	Contact	Edit	*pContactNbr (long)* *sequence number of the contact* *pRelation (string)* *set to this relationship code*	Sets the relationship code on the supplied contact, on the current CAP
convertDate (thisDate)	1.6	Utility	Utility	*thisDate (scriptDateTime)*	*converts a scriptDateTime date to a javascript Date*
convertStringToPhone (theString)	1.6	Utility	Utility	*theString (string)* *string containing information to convert*	Converts the string to phone codes (A=1, D=3, etc), useful with the setIVR function.
copyAddresses (capIdFrom, capIdTo)	1.4	Address	Copy	*capIdFrom (CapIDModel)* CapID object for application to be copied from. *capIdTo (CapIDModel)* CapID object for application to be copied to. If this is ***null*** or omitted, current application is used.	Copies all property addresses from application *capIdFrom* to application *capIdTo*. If application *capIdTo* has a primary address, any primary address in *capIdFrom* will be made non-primary when copied over. Hint: getApplication( ), getParent( ), createChild(), createCap() functions each returns a CapID object.
copyAppSpecific (capId)	1.3	ASI	Copy	*capId (CapIDModel)* CapID object for application where all app spec info fields are to be copied to.	Copies all app spec info fields from current application to the application whose capId object is *capId*. If the target application does not have the same app specific info field, the field is not copied.
copyASIFields (sourceCapId, targetCapId, [ignore$_1$, … ignore$_n$])	1.5	ASI	Copy	*sourceCapId (CapIDModel)* CapID object of CAP to copy from. *targetCapId (CapIDModel)* CapID object of CAP to copy to. *ignore$_1$ to ignore$_n$ (string)(optional)* ASI fields to ignore during the copy.	Copy all ASI fields from the *sourceCapId* CAP to the *targetCapId* CAP with the exception of the fields listed in *ignore$_1$ . . . ignore$_n$*
copyCalcVal (capIdTo)	1.4	Fee	Copy	*capIdTo (CapIDModel)* CapID object of target application.	Copies the calculated job value from the current application to the application whose CapID object is *capIdTo*.

copyConditions (capId)	1.3	Condition	Copy	*capId (CapIDModel)*
				CapID object for application that conditions are to be copied from.
					Copies all conditions from application *capId* to the current application.
					
					Example
					`true ^ subdivapp = getApplication(lookup("SubdivisionXref",{SubDiv})) ;`
					`copyConditions(subdivapp)`

copyConditionsFrom Parcel (parcelNum)	1.4	Condition	Copy	*parcelNum (string)* Parcel number of source parcel.	Copies conditions from the reference parcel *parcelNum* and adds them as conditions to the current application (not to parcels on the current application).

Function	Ver	Category	Type	Parameters	Description
copyContacts (capIdFrom, capIdTo)	1.3	Contact	Copy	*capIdFrom (CapIDModel)* CapID object for application to be copied from. *capIdTo (CapIDModel)* CapID object for application to be copied to. If this is omitted, current application is used.	Copies all contacts from application *capIdFrom* to application *capIdTo*. Note: if target application has a primary contact and the source application also has a primary contact, the target application will end up with 2 primary contacts. Hint - **capId** is the Cap ID object for the current application. - getApplication( ), getParent( ), createChild(), createCap() functions each return a Cap ID object.
copyFees (sourceCapId, targetCapId)	1.5	Fee	Copy	*sourceCapId (capId model)* capID object for CAP from which fees are copied. *targetCapId (capid model)* capID object for CAP to which fees are copied.	Copies all fees from CAP *sourceCapId* to CAP *targetCapId.* Excludes voided or credited fees.
copyLicensedProf (sourceCapId, targetCapId)	1.6	Professional	Copy	*sourceCapId (capId model)* capID object for CAP from which LP are copied. *targetCapId (capid model)* capID object for CAP to which LP are copied.	Copies all licensed professionals from sourceCapID to CAP targetCapId
copyOwner (sourceCapId, targetCapId)	1.6	Contact	Copy	*sourceCapId (capId model)* capID object for CAP from which Contacts are copied. *targetCapId (capid model)* capID object for CAP to which Contacts are copied.	Copies a contacts from sourceCapID to CAP targetCapId
copyParcelGisObjects ()	1.3	GIS	Add	*(none)*	Copies parcel GIS objects to the application.
copyParcels (capIdFrom, capIdTo)	1.4	parcel	Add	*capIdFrom (CapIDModel)* Cap ID object for application to be copied from. *capIdTo (CapIDModel)* Cap ID object for application to be copied to. If this is omitted, current application is used.	Copies all parcels from application *capIdFrom* to application *capIdTo*. Parcel attributes are also copied. Hint - **capId** is the CapID object for the current application. - getApplication( ), getParent( ), createChild(), createCap() functions each return a CapID object.

Function	Ver	Category	Type	Parameters	Description
copySchedInspections (capIdFrom, capIdTo)	1.4	Inspection	Add	*capIdFrom (CapIDModel)* CapID object for application to be copied from. *capIdTo (CapIDModel)* CapID object for application to be copied to. If this is omitted, current application is used.	Copies all scheduled inspections from application *capIdFrom* to application *capIdTo*. Includes inspections that have a pending-type result, but will copy status over as "Scheduled". Inspection type does not need to be on target application to be copied over. Inspection will be copied even if a duplicate scheduled inspection is already on the target application. Hint - **capId** is the CapID object for the current application. - getApplication( ), getParent( ), createChild(), createCap() functions each return a CapID object.
countActiveTasks (wfProcess)	1.4	workflow	Get	*wfProcess (string)* Process name of workflow.	Returns the number of active tasks in the workflow whose process name is *wfProcess*.
countIdentical Inspections ()	1.4	Inspection	Get	*(none)*	Returns the number of inspections that have the same inspection description and status (or result) as the inspection in the current event. Use this function only with the following events: - InspectionResultSubmitAfter - InspectionScheduleAfter - InspectionScheduleBefore
createCap (appType, appName)	1.4	Record	Add	*appType (string)* Four level application type. Must contain 3 slash (/) characters. Do not add spaces before or after slashes. *appName (string)* Application Name.	Creates an application of type *appType* with the application name of *appName.* Returns the new application's CapID object.
createCapComment (vComment, capId optional)	1.6	Record	Add	*vComment (string)* *comment to add* *capId (CapIDModel) (optional)*	Creates a CAP comment for the specified CAP
createChild (grp, typ, stype, cat, appName)	1.3	hierarchy/ related CAPs	Add	*grp (string)* App Group. Top classification of the application *typ (string)* App Type. Second classification of the application *stype (string)* App SubType: 3rd Classification of the application *cat (string)* App Category: 4th Classification of the application *appName (string)* Application name.	Creates an application of type *grp/typ/stype/cat* with the application name of *appName,* and links it as a child to the current application's hierarchy. The following data are copied from the current application to the new child application. parcels contacts property addresses Returns the new child application's capId object, to be used in other functions.

Function	Ver	Category	Type	Parameters	Description
createPublicUserFromContact (contactType optional)	1.6	Contact	Add	*contactType (optional string)* *the public user will be based on this contact type, default is "Applicant"*	Creates a public user account (Accela Citizen Access) with information based on the contact. Useful for automatically creating an online account for applicants that apply in the office. • Creates the public user record • Assigns to current agency • Activates for the current agency • Issues a password reset to their email address • Sends activation email

createRefContactsFromCapContactsAndLink
(pCapId,
contactTypeArray,
ignoreAttributeArray,
replaceCapContact,
overwriteRefContact,
refContactExists)
	1.6	Contact	Add	*pCapId (capIDModel)*
				CAP to work with
				contactTypeArray (array)
				contact types to process, or null for all
				ignoreAttributeArray (array)
				an array of attributes to ignore when creating a REF contact, or null
				replaceCapContact (Boolean)
				not implemented
				overwriteRefContact (Boolean)
				if true, will refresh linked ref contact with CAP contact data
				refContactExists (function)
				function used to determine if the reference contact exists.
					This function can be used as the basis for maintaining a contact-centric database within Accela Automation.
					
					`iArr = new Array();`
					`iArr.push("Partner Percent")`
					`createRefContactsFromCapContactsAndLink(capId,null,iArr,false,true,comparePeopleGeneric);`
					
					In this example, when this code is executed, the function will loop through all contacts on the current CAP. If the contact was hand-entered (not selected and validated from reference contacts) the reference contacts will be searched for a match using the comparePeopleGeneric function. If a match is found, the CAP contact will be "linked" to the reference contact. Also, the reference contact will be "refreshed" with data from the cap contact. All attributes will be refreshed except for the "Partner Percent" field.

Function	Ver	Category	Type	Parameters	Description
createRefLicProf (stateLicNum, licType, [contactType])	1.4	Professional	add, edit	*stateLicNum (string)* State license number. *licType (string)* License type. *contactType (string)(optional)* Contact type.	Creates a new reference Licensed Professional from the Contact on the current CAP whose contact type is *contactType*. The Licensed Professional will have the state license # of *stateLicNum* and license type of *licType*. If a reference Licensed Professional with state license # *stateLicNum* already exists, it will be updated with data from the Contact. Note - Contact's State field must be populated for the Licensed Prof to be created. - The Contact's middle name and address line 3 will not be copied to the Licensed Prof. - If available, the following app specific info fields will be copied to the Licensed Prof (field labels must match exactly): Insurance Co Insurance Amount Insurance Exp Date Policy # Business License # Business License Exp Date
createRefLicProfFrom LicProf ()	1.4	Professional	add, edit	*(none)*	Retrieves the first licensed professional on the application and creates a reference licensed professional record. If a reference record already exists for this licensed professional, updates the reference licensed record with the licensed professional's data from the application.
dateAdd (date, numDays, [workDays])	1.3	utility	Get	*date (string)* Starting date, in format "MM/DD/YYYY" (or any string that will convert to JS date). If null is used, *date* will be the current date. *numDays (integer)* Number of days to add to *date*. Use negative number (e.g. –20) to subtract days from *date*. *workDays (string) (optional)* 'Y' if *numDays* workdays should be added to *date*. Omit if *numDays* calendar days should be added to *date*.	Returns date that results from adding *numDays* days to *date,* as a string in "MM/DD/YYYY" format. Note: Does not work if *date* is wfDate. Will return **NaN/NaN/NaN.**

Function	Ver	Category	Type	Parameters	Description
dateAddMonths (baseDate, numMonths)	1.4	utility	Get	*baseDate (string)* Starting date, in format "MM/DD/YYYY" (or any string that will convert to JS date). If null is used, *date* will be the current date. *numMonths (integer)* Number of months to add to *baseDate*. Use negative number (e.g. -12) to subtract months from *date*.	Returns date that results from adding numMonths months to *baseDate*, as a string in "MM/DD/YYYY" format. If *baseDate* is the last day of the month, the returned date will also be on the last day of the month. If *baseDate* is not the last day of the month, the new date will have the same day of month, unless such a day doesn't exist in the new month (e.g. if *baseDate* is 1/30/2007 and the returned month is February), in which case the new date will be on the last day of the month. Note: Does not work if *baseDate* is wfDate. Will return **NaN/NaN/NaN.**
dateNextOccur (month, day, baseDate, [oddEven])	1.3	utility	Get	*month (string)* Month of new date, as 2-digit month. *day (string)* Day of new date, as 2-digit day. *baseDate (string)* Date from which new date will be determined. In format MM/DD/YYYY or YYYY-MM-YY as used by wfDate variable. *oddEven (string) (optional)* Specifies if the new date should be in an odd or even year. Enter "odd" or "even".	Returns the next occurence of *month* and *day* after *baseDate*. If *oddEven* is "odd", gets the next occurence of *month* and *day* after *baseDate* in an odd year (i.e., year is an odd number). If *oddEven* is "even", gets the next occurrence of *month* and *day* after *baseDate* in an even year. *baseDate* can be a date string in MM/DD/YYYY format, or an event-specific variable (e.g. wfDate) whose date format is YYYY-MM-DD.
deactivateTask (wfTask, [wfProcess])	1.6	Workflow	Edit	*wfTask (string)* Workflow task to be deactivated *wfProcess (string) (optional)* Process name of workflow task *wfTask.*	Deactivates the task, similar to setting Active? = N in the workflow supervisor portlet
deleteTask (targetCapId, deleteTaskName)	1.6	Workflow	Edit	*targetCapId (capID Model)* *CAP to affect* *deleteTaskName (string)* *name of task to delete*	Permanently removes the named task from the workflow. The task will no longer appear.
editAppName (newname, [capId])	1.5	Record	Edit	*newName (string)* New application name *capId (CapIDModel) (optional)* CapId object for application	Updates application name to *newName*. Returns **true** if successful or **false** if update fails.

Function	Ver	Category	Type	Parameters	Description
editAppSpecific (itemName, itemValue, [capId])	1.3	ASI	Edit	*itemName (string)* App Specific Info field to edit . *itemValue (string)* Value that the app spec info field *itemName* should be changed to. *capId (CapIDModel) (optional)* CapID object for application whose app spec info field *itemName* is to be changed to *itemValue*.	Updates the value of the app specific info field *itemName* with the value *itemValue*. Also updates the internal list of values, so that future criteria/action pairs will see the correct value. If no *capId* is supplied, then the current CAP is used.
editBuildingCount (numBuild, [CapId])	1.6	Record	Edit	*numBuild (string)* *new number of buildings* *capId (optional capIDmodel)* *capID to affect*	Edits the building count on the CAP detail.
editContactType (existingType, newType, [capId])	1.5	Contact	Edit	*existingType (string)* Existing Contact Type *newType (string)* New Contact Type *capId (CapIDModel) (optional)* Capid Object	Updates Contact Type for all contacts on a CAP to newtype when the existing Contact Type is equal to the existingType. Hint: getApplication( ), getParent( ), createChild( ) functions each returns a capId object that can be used in the capId parameter
editHouseCount (numHouse, [capId])	1.5	Record	Edit	*numHouse (string)* New house count *capId (CapIDModel) (optional)* CapId object for application	Updates the CAP's house count field to *numHouse*. Returns **true** if successful or **false** if update fails.
editInspectionRequiredFlag (inspType, reqFlag, [capId])	1.6	Inspection	Edit	*inspType (string)* *inspection type to edit* *reqFlag (Boolean)* *if true, sets the required flag to "Y", otherwise "N"* *capId (optional)* *target Cap Id*	Sets the inspection milestone flag 'Inspection Required" to Y or N
editLookup (stdChoice, stdValue, stdDesc)	1.5	utility	add/ edit	*stdChoice (string)* Name of standard choice. *stdValue (string)* Name of standard choice value. *stdDesc (string)* New standard choice description.	Attempts to find existing standard choices value called *stdValue* in the standard choices item called *stdChoice.* If found, updates the existing Value Description for *stdValue*. If *stdValue* is not found, adds the new value *stdValue* with the Value Desc of *stdDesc*.
editPriority (priority, [capId])	1.5	Record	edit	*priority (string)* New priority *capId (CapIDModel) (optional)* CapId object for application	Updates the CAP's Priority field to *priority*. Returns **true** if successful or **false** if update fails.
editRefLicProfAttribute (pLicNum, pAttributeName, pNewAttributeValue)	1.6	Professional	Edit	*pLicNum (string)* License number of reference LP *pAttributeName* *Label of the attribute to update* *pNewAttributeValue* *New attribute value*	Updates the attribute (template data) on a reference licensed professional record

Function	Ver	Category	Type	Parameters	Description
editReportedChannel (reportedChannel, [capId])	1.5	Record	edit	*reportedChannel (string)* New reported channel value *capId (CapIDModel) (optional)* CapId object for application	Updates the CAP's Reported Channel field to *reportedChannel*. Returns **true** if successful or **false** if update fails.
editScheduledDate (scheduledDate, [capId])	1.6	Inspection	Edit	*scheduledDate (string)* New schedule date value *[capId] (optional capId)* *capId to modify*	Edits the schedule date in CAP detail on the selected CAP
editTaskComment (wfTask, wfComment, [wfProcess])	1.3	workflow	edit	*wfTask (string)* Workflow task whose comment should be updated. *wfComment (string)* Comment to be given to *wfTask*. *wfProcess (string) (optional)* Process name of workflow task *wfTask.*	Adds the status comment *wfComment* to workflow task *wfTask*. If *wfTask* has an existing comment, the comment will be replaced by *wfComment*. *wfTask* does not have to be active. Status date is not updated. No workflow history record is created. If application's workflow contains duplicate *wfTask* tasks, use parameter *wfProcess* to specify the process or subprocess whose *wfTask* should be edited.
editTaskDueDate (wfTask, wfDate, [wfProcess])	1.3	workflow	edit	*wfTask (string)* Workflow task. *wfDate (string)* Due date to be given to *wfTask*. *wfProcess (string) (optional)* Process name of workflow task *wfTask.*	Sets the due date of the workflow task *wfTask* to *wfDate*. If *wfTask* is "*", sets due dates on all workflow tasks on the application. No workflow history record is created. If application's workflow contains duplicate *wfTask* tasks, use parameter *wfProcess* to specify the process or subprocess whose *wfTask* should be edited.
editTaskSpecific (wfTask, itemName, itemValue, [capId])	1.3	workflow	edit	*wfTask (string)* Workflow task. *itemName (string)* Task Specific Info field to edit . *itemValue (string)* Value that the task spec info field *itemName* should be changed to. *capId (CapIDModel) (optional)* CapID object for application whose task spec info field *itemName* is to be changed to *itemValue*.	Updates the value of the task specific info field *itemName* for workflow task *wfTask* to the value *itemValue*. Also updates the internal list of values, so that future criteria/action pairs will see the correct value. If *capId* is supplied, updates the specified task specific info field on the application whose CapID object is *capId*.
email (toEmail, fromEmail, subject, text)	1.4	utility	Utility	*toEmail (string)* Email address of recipient. *fromEmail (string)* Email address of sender. *subject (string)* text that appears in subject line of email *text (string)* text that appears in body of email	Sends an email to the email address *toMail* from the email address *fromMail*. The email's subject line is *subject* and its content is *text*.

Function	Ver	Category	Type	Parameters	Description
emailContact (subject, text, [contactType])	1.3	Contact	Utility	*subject (string)* text that appears in subject line of email *text (string)* text that appears in body of email *contactType (string) (optional)* Contact Type that email is sent to. Default is "Applicant".	Sends an email to the contact on the current application whose Contact Type is *contactType*. Uses the email address in the contact screen. Default contact is "Applicant". Example `inspResult.equals("Passed") ^ emailContact("Inspection Results", "Your inspection " + inspType + " has passed.", "Contractor")`
endBranch	1.6	Utility	Utility	*None*	Immediately stops execution of the branch (standard choice) that is currently executing. Script controls will continue executing from the calling standard choice, if any. For example: 01 true ^ endBranch() 02 true ^ comment("this will not execute")
executeASITable (tableArray)	1.5	ASI	execute	*tableArray (array)* Application specific info table array	Executes an ASI table as if it were script commands. No capability for else or continuation statements. Assumes that there are at least three columns named "Enabled", "Criteria", and "Action". Will replace token in the controls.
exists (eVal, eArray**)**	1.6	Utility	True/False	*eVal* The search value *eArray (array of strings)* *potential matches*	Searches the array *eArray* for the value *eVal.* Returns true if the value is found in the array. Example; Values = new Array("Apple","Pear","Banana"); X = exists("Apple",Values); X is true.

Function	Ver	Category	Type	Parameters	Description
externalLP_CA_3_2 (licNum, rlpType, doPopulateRef, doPopulateTrx, itemCap)	1.6	Professional	Utility	*licNum (string)* *Valid CA license number. Non-alpha, max 8 characters. If null, function will use the LPs on the supplied CAP ID* *rlpType(string)* *License professional type to use when validating and creating new LPs* *doPopulateRef (Boolean)* *If true, will create/refresh a reference LP of this number/type* *doPopulateTrx(Boolean)* *If true, will copy create/refreshed reference LPs to the supplied Cap ID. doPopulateRef must be true for this to work* *itemCap (capId)* *If supplied, licenses on the CAP will be validated. Also will be refreshed if doPopulateRef and doPopulateTrx are true*	Validates a license with the California State License Board and refreshes LP information with results. See the "CSLB Inteface using the externalLP_CA function - v3_0.pdf" document for detailed information. Examples: appsubmitbefore (will validate the LP entered, if any, and cancel the event if the LP is inactive, cancelled, expired, etc.) cslbMessage = externalLP_CA(CAELienseNumber,false,false,CAELienseType,null); appsubmitafter (update all CONTRACTOR LPs on the CAP and REFERENCE with data from CSLB. Link the CAP LPs to REFERENCE. Pop up a message if any are inactive...) cslbMessage = externalLP_CA(null,true,true,"CONTRACTOR",capId)
feeAmount (feeCode, [fStatus$_1$, … fStatus$_n$])	1.5	Fee	Get	*feeCode (string)* Fee code. *fStatus$_1$ … fStatus$_n$ (string) (optional)* List of fee statuses to check for. Enter one or more statuses.	Returns the total amount of the all fees on the application whose fee code is *feeCode.* If optional *fStatus$_1$ … fStatus$_n$* parameter(s) are supplied, also checks that *feeCode* has one of the statuses in *fStatus$_1$ … fStatus$_n$*. Hint: A fee will have one of the following statuses: NEW, INVOICED, VOIDED, CREDITED.
feeBalance (feeCode, [feeSchedule])	1.4	Fee	Get	*feeCode (string)* Fee code. *feeSchedule (string)(optional)* Fee schedule.	Returns the total balance due for all fees on the application whose fee code is *feeCode*. If parameter *feeSchedule* is used, retrieves those fees whose schedule is *feeSchedule*.
feeCopyByDateRange (pStartDate, pEndDate, [feeStatus], [feeStatus])	1.6	Fee	Add	*pStartDate* Starting search date for fee items *pEndDate* Ending search date for fee items *feeStatus (optional string)* Search for fee items of this status *feeStatus (optional string)* Search for fee items of this status	On the current CAP, will search for fees in the given date and status criteria, then copy the fees onto the current CAP.

Function	Ver	Category	Type	Parameters	Description
feeExists (feeCode, [fStatus$_1$, … fStatus$_n$])	1.3	Fee	true/ false	*feeCode (string)* Fee code of fee to check for. *fStatus$_1$ … fStatus$_n$ (string) (optional)* List of fee statuses to check for. Enter one or more statuses.	Returns **true** if a fee whose fee code is *feeCode* has been added to the application. If optional *fStatus$_1$ … fStatus$_n$* parameter(s) are supplied, also checks that *feeCode* has one of the statuses in *fStatus$_1$ … fStatus$_n$*. Hint: A fee will have one of the following statuses: NEW, INVOICED, VOIDED, CREDITED. Example: To determine if fee “FEE001” has been added and not invoiced: `feeExists("FEE001","NEW")`
feeGetTotByDateRange(startDate, endDate, [fStatus$_1$, … fStatus$_n$])	1.3	Fee	Get	*startDate (string)* Start of date range, in format MM/DD/YYYY. *endDate (string)* End of date range, in format MM/DD/YYYY. *fStatus$_1$ … fStatus$_n$ (string) (optional)* List of fee statuses to check for. Enter one or more statuses.	Returns total amount of fees that were assessed during the date range *startDate* to *endDate*. If optional *fStatus$_1$ … fStatus$_n$* parameter(s) are supplied, the fee must have one of the statuses in *fStatus$_1$ … fStatus$_n$*. Hint: A fee will have one of the following statuses: NEW, INVOICED, VOIDED, CREDITED. Note: Fees are retrieved by their initial assess date, not invoiced date.
feeQty (feestr)	1.6	Fee	Get	*feeStr (string)* fee item to search	On the current CAP, returns the quantity field of the given fee item.
getAppIdByASI (itemName, itemValue, appType)	1.4	ASI	Get	*itemName (string)* App specific info field name to search for. *itemValue (string)* App specific info field value to search for. CapID object for application whose app spec info field *itemName* is to be changed to *itemValue*. *appType (string)* Four level application type. Must contain 3 slash (/) characters. Do not add spaces before or after slashes. The asterisk (*) may be used as a wildcard to match all entries for a given level.	Returns the application number (cap ID string) of the first CAP whose application type matches *appType* and whose application specific info field *itemName* has the value of *itemValue*.
getAppIdByName (group, type, appName)	1.3	Record	Get	*group (string)* Application group. *type (string)* Application type. *appName (string)* Application name.	Returns the cap ID string of the first application whose application type begins with *group / type* and whose application name is *appName*. Hint: The parameter *type* is the 2nd value in the 4 level application type.
getApplication (applicationNumber)	1.3	Record	Get	*applicationNumber (string)* Application # (B1_ALT_ID)	Returns the CapID object for application *applicationNumber* that can be used by other functions.

Function	Ver	Category	Type	Parameters	Description
getAppSpecific (itemName, [capId])	1.3	Record	Get	*itemName (string)* Application Specific Info field to get. *capId (CapIDModel) (optional)* CapID object for application.	Returns the value of the application spec info field *itemName*. If capId is supplied, returns the value of *itemName* on the application whose CapID object is *capId*.
getCapByAddress (appType)	1.4	Address	Get	*appType (string)* Four level application type. Must contain 3 slash (/) characters. Do not add spaces before or after slashes. The asterisk (*) may be used as a wildcard to match all entries for a given level.	Returns the first application having the same address as the current application and whose application type matches *appType*, as a CapID object. Addresses are matched by Street # (start), Street Name, Street Direction, Street Suffix, and Zip. The current application may be returned if its application type is appType. If no applications are found, the function does not return any value.
getChildren (appType, [parentCapId], [skipChildCapId])	1.4	hierarchy/ related CAPs	Get	*appType (string)* Four level application type. Must contain 3 slash (/) characters. Do not add spaces before or after slashes. The asterisk (*) may be used as a wildcard to match all entries for a given level. parentCapId *(CapIDModel)(optional)* CapID object for parent application. Use **null** if *skipChildCapId* parameter is used. skipChildCapId *(CapIDModel)(optional)* CapID object of child application to exclude.	Returns all child CAPs whose application type matches *appType,* as an array of CapID objects. If the *parentCapId* parameter is used, returns child CAPs of the application whose CapID object is *parentCapId*. If the *skipChildCapId* parameter is used, function will exclude any child CAP whose CapID object is *skipChildCapId*. See also: **childGetByCapType** function.
getChildTasks (taskName, [capId])	1.6	Workflow		*taskName* *Name of criteria parent task* *capId (optional capId)* *CAP to search*	Will return an array of taskScriptModel objects, which represent the child tasks (sub process) of the criteria task.

Function	Ver	Category	Type	Parameters	Description
getContactArray ([capId])	1.4	Contact	Get	*capIdFrom (CapIDModel) (optional)* CapID object for source application.	Retrieves field values and custom attribute values for all contacts and returns them as an array of associative arrays. Each element in the outer array contains an associative array of values for one contact. Each element in each inner associative array is a different field. The following fields are retrieved: Contact Field — Element Name First Name — firstName Last Name — lastName Business Name — businessName Phone 1 — phone1 Phone 2 — phone2 Contact Type — contactType Contact Relationship — relation Sequence Number — contactSeqNumber All custom attributes are also added to the associative array, where the element name is the attribute name (in upper-case). Note that the attribute name may not be the same as the attribute label. If the parameter *capIdFrom* is used, function retrieves contacts from the application whose CapID object is *capIdFrom*.
getCSLBInfo (updateLicProf, warnExpire)	1.4	Professional	Edit	*updateLicProf (boolean)* Use **true** if the CAP's license professional must be updated with data from the California State License Board (CSLB); otherwise, use **false.** *warnExpire (boolean)* Use **true** if warning message should appear if license has expired; otherwise, use **false**.	Selects the first licensed professional on the CAP and retrieves its data from the California State License Board (CSLB). If *warnExpire* is **true**, shows a warning message if the license has expired. If *updateLicProf* is **true**, updates the CAP's licensed professional with data from CSLB. The following fields are updated: - Business Name - Address Line 1 - Address Line 2 - City - State - Zip - Phone Number - Issued Date - Expiration Date Returns **false** if the CAP has no licensed professional, if the license cannot be found at CSLB, or if any error is encountered.
getDepartmentName (userID)	1.4	utility	Get	*userID (string)* *User's user ID.*	Returns the department of the user whose ID is *userID*.

Function	Ver	Category	Type	Parameters	Description

getGISBufferInfo
(service,
layer,
distance,
[attribute$_1$,
…
attribute$_n$])
	1.4	GIS	Get	*service (string)*
				GIS Service name
				layer (string)
				GIS layer, i.e., object that function is testing proximity to
				distance (integer)
				Distance (in feet) of GIS objects on CAP to attributes in *layer*
				attribute$_1$…attribute$_n$ (strings)(optional)
				Additional attributes of the GIS layer to retrieve.
					Returns an array of associative arrays. Each element in the outer array is a GIS object (from the indicated layer) within the buffer from the CAP's GIS object. Each element in the inner associative array is a requested attribute.
					
					Example
					`x = getGISBufferInfo("NewtonCounty","Parcels","50","NAME1","TOTACRES");`
					
					`x[0]["TOTACRES"] = 0.46`
					`x[0]["NAME1"] = "JENNINGS DEMETRIA C"`
					`x[1]["TOTACRES"] = 0.46`
					`x[1]["NAME1"] = "SIMMS ROCK & VALARIE"`
					`x[2]["TOTACRES"] = 0.46`
					`x[3]["NAME1"] = "PAUL NEVILLE & MARGARET"`

getGISInfo (service, layer, attribute)	1.4	GIS	Get	*service (string)* GIS Service name *layer (string)* GIS layer. *attribute (string)* *Name of attribute to retrieve.*	Returns the attribute value for *attribute* in the GIS layer *layer* for the last GIS object on the CAP. Use with all events (and master scripts) except ApplicationSubmitBefore.
getGISInfoArray (svc, layer, attributename)	1.6	GIS	Get	*service (string)* GIS Service name *layer (string)* GIS layer. *attribute (string)* *Name of attribute to retrieve.*	Similar to getGISInfo, except will return an array of values for the given attribute, instead of the first value found.
getInspector (inspDesc)	1.3	Inspection	Get	*inspDesc (string)* Inspection description.	Returns the user ID of the inspector assigned to inspection *inspDesc*, whether scheduled or completed. If more than one *inspDesc* is on the application, the first inspection found is selected, which may or may not be the *inspDesc* with the earliest inspection date.
getLastInspector (inspDesc)	1.4	Inspection	Get	*inspDesc (string)* Inspection description.	Returns the user ID of the last inspector to result the inspection *inspDesc*.
getLastScheduledInspector (insp2Check)	1.6	Inspection	Get	*Insp2Check (string)* Inspection Description	Returns the user ID of the last inspector to be schedule on the inspection *insp2check*
getLicenseProfessional (itemcapId)	1.6	Professional	Get	*itemCapId (capId)* *capID to use*	Returns an array of LicensedProfessional objects that represent all LPs on the specified CAP
getParent ( )	1.3	hierarchy/ related CAPs	Get	*(none)*	Returns the capId object for the first parent of the current application.

Function	Ver	Category	Type	Parameters	Description
getParents ([appType])	1.5	hierarchy/ related CAPs	Get	*appType (string) (optional)* Four level application type. Must contain 3 slash (/) characters. Do not add spaces before or after slashes. The asterisk (*) may be used as a wildcard to match all entries for a given level.	Returns all parents on the current application in a CapID object array. If *appType* parameter is passed, only returns parent CAPs whose application type matches the *appType* parameter string pattern.
getRefLicenseProf (refstlic)	1.6	Professional	Get	*refstlic (string)* state license number to search for	Returns a reference licensed professional object for the LP that matches the state license number value
getRelatedCapsBy Address (appType)	1.4	Address	Get	*appType (string)* Four level application type. Must contain 3 slash (/) characters. Do not add spaces before or after slashes. The asterisk (*) may be used as a wildcard to match all entries for a given level.	Returns all applications having the same address as the current application and whose application type matches *appType*, as an array of CapID objects. Addresses are matched by Street # (start), Street Name, Street Direction, and Street Suffix. The current application is not included in the returned array. Applications retrieved do not have to be a parent or child of the current application. If no related applications are found, the function does not return any value.
getRelatedCapsBy Parcel (appType)	1.4	parcel	Get	*appType (string)* Four level application type. Must contain 3 slash (/) characters. Do not add spaces before or after slashes. The asterisk (*) may be used as a wildcard to match all entries for a given level.	Returns all applications having the same parcel as the current application and whose application type matches *appType*, as an array of CapID objects. The current application is not included in the returned array. Applications retrieved do not have to be a parent or child of the current application. If no related applications are found, the function does not return any value.
getReportedChannel ([capId])	1.5	Record	Get	*capId (CapIDModel) (optional)* CapID object for application.	Returns the value of the Reported Channel field as a string. If the Reported Channel field is null, an empty string is returned.
getScheduledInspId (insp2Check)	1.6	Inspection	Get	*insp2Check* *Inspection description*	Returns the internal sequence number for the inspection record that matches the description. Will only return values for "Scheduled" inspections, not resulted inspections. The sequence number that is returned can be used with other functions, such as autoAssignInspection
getShortNotes ([capId])	1.5	Record	Get	*capId (CapIDModel) (optional)* CapID object for application.	Returns the value of the Short Notes field as a string. If the Short Notes field is null, an empty string is returned.
getTaskDueDate (wfTask, [wfProcess])	1.6	Workflow	Get	*wfTask (string)* Workflow task name. *wfProcess (string) (optional)* Workflow process name.	Returns the due date (string) of the requested workflow task on the current CAP. If application's workflow contains duplicate *wfTask* tasks, use parameter *wfProcess* to specify the process or subprocess whose *wfTask* should be checked. Hint: *wfProcess* is R1_PROCESS_CODE in the GPROCESS and SPROCESS tables. *wfProcess* is normally in uppercase.

Function	Ver	Category	Type	Parameters	Description

getTaskStatusForEmail
(wfProcess)
	1.3	workflow	Get	*wfProcess (string)*
				Process name of workflow.
					Gets all completed tasks on workflow *wfProcess* and returns their task name, status, and comments (if any) in the following format:
					`Task Name: {task name}`
					`Task Status: {task status]`
					`Task Comments: {status comments}`
					The above block is repeated for each completed task.

inspCancelAll ()	1.4	Inspection	Edit	*(none)*	Cancels all scheduled and incomplete inspections on the current application. Returns **true** if at least one inspection is cancelled; otherwise, returns **false.**
invoiceFee (fcode, fperiod)	1.5	Fee	Edit	*fcode (string)* Fee code of the fee to be invoiced. *fperiod (string)* Fee period of the fee to be invoiced.	Invoices all assessed fees with fee code of *fcode* and fee period of *fperiod*. Returns **true** if assessed fee is found, else returns **false**.
isScheduled *(inspDesc)*	1.3	Inspection	true/ false	*inspDesc (string)* Inspection description.	Returns **true** if the inspection *inspDesc* has been scheduled or resulted for the current application. Note: To determine if an inspection is scheduled but not yet resulted, use the checkInspectionResult function and use 'Scheduled' for the *inspResult* parameter.
isTaskActive (wfTask, [wfProcess])	1.3	workflow	true/ false	*wfTask (string)* Workflow task name. *wfProcess (string) (optional)* Workflow process name.	Returns **true** if workflow task *wftask* is active, or **false** if it is not. If application's workflow contains duplicate *wfTask* tasks, use parameter *wfProcess* to specify the process or subprocess whose *wfTask* should be checked. Note: If used with the WorkflowTaskUpdateAfter event, this function will return **true** if *wfTask* becomes active as a result of the WorkflowTaskUpdateAfter event. It returns **false** if *wfTask* becomes inactive as a result of the WorkflowTaskUpdateAfter event. Hint: *wfProcess* is R1_PROCESS_CODE in the GPROCESS and SPROCESS tables. *wfProcess* is normally in uppercase.

Function	Ver	Category	Type	Parameters	Description
isTaskComplete (wfTask, [wfProcess])	1.3	workflow	true/ false	*wfTask (string)* Workflow task name. *wfProcess (string) (optional)* Workflow process name.	Returns **true** if workflow task *wftask* is completed, or **false** if it is not. If application's workflow contains duplicate *wfTask* tasks, use parameter *wfProcess* to specify the process or subprocess whose *wfTask* should be checked. Note: If used with the WorkflowTaskUpdateAfter event, this function will return **true** if *wfTask* becomes completed as a result of the WorkflowTaskUpdateAfter event. Hint: *wfProcess* is R1_PROCESS_CODE in the GPROCESS and SPROCESS tables. *wfProcess* is normally in uppercase.
isTaskStatus (wfTask, wfStatus [wfProcess])	1.3	workflow	true/ false	*wfTask (string)* Workflow task name. *wfStatus (string)* Workflow status. *wfProcess (string) (optional)* Workflow process name.	Returns **true** if workflow task *wftask* has the current status of *wfStatus*, or **false** if it does not. Returns **false** if *wftask* is not found. If application's workflow contains duplicate *wfTask* tasks, use parameter *wfProcess* to specify the process or subprocess whose *wfTask* should be checked. Hint: *wfProcess* is R1_PROCESS_CODE in the GPROCESS and SPROCESS tables. *wfProcess* is normally in uppercase.
jsDateToASIDate (jsDate)	1.5	utility	Get	*jsDate(JavaScript Date)* JavaScript Date object	Converts the JavaScript Date object to a string with a zero pad date format that can be used in ASI, TSI, and ASI Table date fields.
jsDateToMMDDYYYY (jsDate)	1.4	utility	Get	*jsDate (JavaScript Date)* JavaScript Date object.	Converts the JavaScript Date object *jsDate* to a string in the format MM/DD/YYYY. Returns the date as a string in the format MM/DD/YYYY. Hint: Use this function if you wish to display a JavaScript date in the format MM/DD/YYYY. The result of this function should not be used directly to compare against another date.

Function	Ver	Category	Type	Parameters	Description
licEditExpInfo (expStatus, expDate)	1.4	renewal	Edit	*expStatus (string)* Expiration status. Use **null** if only expiration date is to be edited. *expDate (string)* Expiration date. Use **null** if only expiration status is to be edited.	Changes the CAP's expiration status to *expStatus* and expiration date to *expDate*. If *expStatus* is null, expiration status will not be changed. If *expDate* is null, expiration date will not be changed. Use function with license CAPs only, i.e., Application type begins with "Licenses". Note: - *expDate* can be in YYYY-MM-DD or MM/DD/YYYY format. - Script will throw error if CAP does not have Renewal Info.
licenseObject (licNumber, [capId])	1.6	Renewal	Utility	*licNumber (string)* state license number of the reference licensed professional to be linked to this license object. capId (optional capId) capId to use for the license object	This function creates a helper object that is used to view and modify license and expiration information. If licNumber has a value, the helper object will attempt to replicate changes to a reference license professional, as well as the CAP. Methods: .setExpiration (date string) will set the expiration date of the CAP, and the LP if linked .setIssued (date string) will set the issued date of the LP if linked. .setLastRenewal (date string) will set the last renewal date of the LP if linked. .setStatus (status string) will set the expiration status of the CAP .getStatus() will return the expiration status of the CAP .getCode() will get the expiration code of the CAP
loadAddressAttributes (attrArray, [capId])	1.6	Address	Get	*attrArray (array)* *target array of address attributes* *capId (optional capId)* *capID to search*	Will populate attrArray as a associate array of address attributes and values based on the address associated with the CAP

Function	Ver	Category	Type	Parameters	Description
loadAppSpecific (asiArray, [capId])	1.4	ASI	Get	*asiArray (array)* Associative array. *capId (CapIDModel) (optional)* CapID object for application where all app spec info fields are to be copied from.	Retrieves all application specific info fields and adds them to the associative array *asiArray.* The element name is the application specific info field name and the element value is the field value. If the user configurable variable **useAppSpecificGroupName** on the master script is set to **true**, the group name is appended to the beginning of the field name with a period, e.g. "CONSTRUCTION_INFO.Construction Type". Application specific info table data is not retrieved. If parameter capId is used, function will retrieve application info fields from the application whose CapID object is *capId*.
loadASITable (tableName, [capId])	1.6	ASI	Get	*tableName (string)* *name of the ASI table to load* *capId (optional cap id)* *CAP id to load the table from*	Returns an array of associate arrays that contain objects representing the contents of the ASI table for the selected CAP. The underlying object is an "asiTableValObj" that contains three properties: .fieldValue = value of the table .columnName = name of the column for this value .readOnly = "Y" if the field is read only, "N" if not. For example: myTable = loadASITable("EXAMPLE TABLE") firstRow = myTable[0]; columnA = firstRow["Column A"] columnB = firstRow["Column B"] comment("value of column a is : " + columnA.fieldValue) comment("column a read only property is : " + columnA.readOnly) The fieldValue property of the asiTableValObj object is the default property, so the following will also work: comment("value of column a is : " + columnA);

Function	Ver	Category	Type	Parameters	Description

loadASITables
([capId])
	1.6	ASI	Get	*capId (optional cap id)*
				CAP id to load the tables from
					Similar to the loadASITable function, except global variables are created for each ASI table on the requested CAP.
					
					The names of the tables may be edited to remove whitespace and leading digits, so that they become appropriate JavaScript variables.
					
					For example:
					
					loadASITables();
					
					if (typeof(PROPERNAMES) == “object”)
					comment(“number of rows in the ‘PROPER NAMES’ table : “ + PROPERNAMES.length)
					
					variables are not created for tables that do not have any data, so you must first check for the presence of the table variable by using the JavaScript typeof operator as shown above.
					
					loadASITables is executed by default in all master scripts.

<table>
<tr><th>Function</th><th>Ver</th><th>Category</th><th>Type</th><th>Parameters</th><th>Description</th></tr>
<tr><td>loadFees
([capId])</td><td>1.5</td><td>Fee</td><td>Get</td><td>capId (CapIdModel) (optional)
CapID object of CAP to load fees from.</td><td>Retrieves all assessed fees for the CAP capId and returns them as an array of associative arrays. Each element in the outer array contains an associative array of values for one fee. Each element in each inner associative array is a different field. The following fields are retrieved:
<table><tr><th>Fee Field</th><th>Element Name</th></tr><tr><td>Sequence Num</td><td>sequence</td></tr><tr><td>Fee Code</td><td>code</td></tr><tr><td>Description</td><td>description</td></tr><tr><td>Unit</td><td>unit</td></tr><tr><td>Amount</td><td>amount</td></tr><tr><td>Amount Paid</td><td>amountPaid</td></tr><tr><td>Applied Date</td><td>applyDate</td></tr><tr><td>Effective Date</td><td>effectDate</td></tr><tr><td>Status</td><td>status</td></tr><tr><td>Received Date</td><td>redDate</td></tr><tr><td>Fee Period</td><td>period</td></tr><tr><td>Display Order</td><td>display</td></tr><tr><td>Account Code 1</td><td>accCodeL1</td></tr><tr><td>Account Code 2</td><td>accCodeL2</td></tr><tr><td>Account Code 3</td><td>accCodeL3</td></tr><tr><td>Fee Formula</td><td>formula</td></tr><tr><td>Sub Group</td><td>subGroup</td></tr><tr><td>Calculation Flag</td><td>calcFlag</td></tr></table></td></tr>
<tr><td>loadGuideSheetItems
(inspId)</td><td>1.6</td><td>Inspection</td><td>Get</td><td>inspID (long)
inspection sequence number to load</td><td>Returns an associative array of guide sheet items from the indicated inspection.

For example:

gsArray = loadGuideSheetItems(234323);

comment(gsArray[“Privacy Violation”])

will display the value of the ”Privacy Violation” guide sheet item.</td></tr>
</table>

Function	Ver	Category	Type	Parameters	Description
loadParcelAttributes (parArray, [capId])	1.4	parcel	Get	*parArray (array)* *Associative array.* *capId (CapIDModel) (optional)* CapID object for application where parcel attributes are to be copied from.	Retrieves all parcel fields (including custom attributes) and adds them to the associative array *parArray.* The element name is the field name (prefixed with "ParcelAttribute.") and the element value is the field value. The following standard parcel fields are included: - ParcelAttribute.Block - ParcelAttribute.LegalDesc - ParcelAttribute.Book - ParcelAttribute.Lot - ParcelAttribute.CensusTract - ParcelAttribute.MapNo - ParcelAttribute.CouncilDistrict - ParcelAttribute.MapRef - ParcelAttribute.ExemptValue - ParcelAttribute.ParcelStatus - ParcelAttribute.ImprovedValue - ParcelAttribute.SupervisorDistrict - ParcelAttribute.InspectionDistrict - ParcelAttribute.Tract - ParcelAttribute.LandValue - ParcelAttribute.PlanArea If the CAP has multiple parcels, only fields for the last parcel will be retrieved. If parameter capId is used, function will retrieve parcel fields from the application whose CapID object is *capId*.
loadTasks (applicationNumber)	1.3	workflow	Get	*applicationNumber (string)* Application # (B1_ALT_ID)	Returns an array of workflow task objects for the application *applicationNumber.*
loadTaskSpecific (asiArray, [capId])	1.4	workflow	Get	*tsiArray (array)* *Associative array.* *capId (CapIDModel) (optional)* CapID object for application where all task spec info fields are to be copied from	Retrieves all task specific info fields and adds them to the associative array *tsiArray.* The element name is the task specific info field name and the element value is the field value. If the user configurable variable **useTaskSpecificGroupName** on the master script is set to **true**, the workflow process code and task name are prepended to the field name, e.g. "BLDGPROCESS.Application Submittal.Date Received". If parameter *capId* is used, function will retrieve task specific info fields from the application whose CapID object is *capId*.

Function	Ver	Category	Type	Parameters	Description

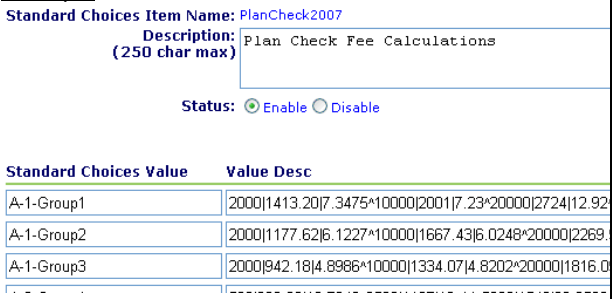
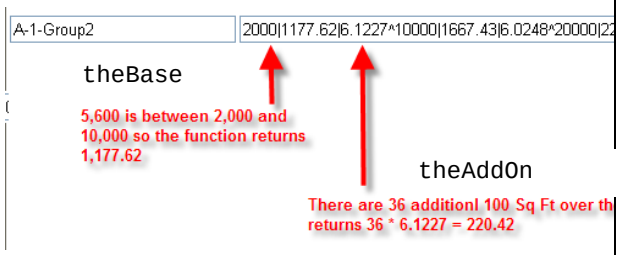
logDebug
(debugVal,
[debugLevel])
	1.6	Utility	Utility	*debugVal*
				Value to be displayed on the debug window
				debugLevel
				indicates debug content destination
					Displays debug information, depending on the showDebug global variable setting. debugLevel will override this setting for this message only.
					
					debugLevel = false // no output
					debugLevel = 1 // screen output
					debugLevel = 2 // output to biz server log
					debugLevel = 3 // output to screen and biz log

lookup (itemName, valueName)	1.3	utility	Get	*itemName (string)* Standard Choices Item Name *valueName (string)* Standard Choices Value	looks up *valueName* in Standard Choices Item *itemName,* and returns its value description. Essentially uses standard choices as a lookup table. Returns the Value Desc corresponding to the Standard Choices Value *valueName* in the Standard Choices Item *itemName.* If *valueName* is not found, returns **undefined**.

Function	Ver	Category	Type	Parameters	Description

lookupDateRange (itemName, compareDate, [valueIndex])	1.4	utility	Get	*itemName (string)*
				Item Name of Standard Choices used as lookup table
				compareDate (string)
				Date that determines which row to return. Use string in format MM/DD/YYYY, e.g. "07/21/2000"
				valueIndex (integer)(optional)
				Determines which value is returned. Defaults to 1, the first value.
					Matches *compareDate* against a series of dates in the Standard Choices called *itemName*. If *compareDate* falls on or after date 1 but but before date 2, returns the value following the caret (^) on date 1's right. If parameter *valueIndex* is used, returns the value immediately after the *valueIndex*'th caret (^) following the matching date.
					
					Set up the **Standard Choices** lookup table as follows:
					Value column = Four digit incremental index. Must be left zero padded to four digits. Entire table must be consecutive.
					Value Desc column = at least two values separated with the caret (^) symbol. First value is effective date (MM/DD/YYYY format), remaining values are returned by the function.
					
					<u>Example</u>
					
					 
					
					`lookupDateRange("test date lookup","5/5/2002")` returns 33333
					`lookupDateRange("test date lookup","1/5/2000",2)` returns 12222
					`lookupDateRange("test date lookup","1/1/2010")` returns 44444
					`lookupDateRange("test date lookup","1/1/1999")` returns **undefined** since there is no entry effective for that date.
					`lookupDateRange("test date lookup","1/5/2000",3)` returns **undefined** since there are not 3 values.
					
					Sample script controls:
					`01 appMatch("Building/Residential/SFD/*") ^lookupIndex=1`
					`02 appMatch("Building/Residential/Duplex/*") ^ lookupIndex = 2`
					`03 true ^ addFee("FEECODE","FEESCHED","FEEPERIOD", lookupDateRange("test date lookup", filedate,`

Function	Ver	Category	Type	Parameters	Description

lookupFeesByValuation (itemName, valueName, compareValue, [valueIndex])	1.4	Fee	Get	*itemName (string)*
				Item Name of Standard Choices used as lookup table
				valueName (string)
				Standard Choices Value
				compareValue (number)
				Number value (e.g. valuation) to be compared.
				valueIndex (integer)(optional)
				Determines which value is returned. Defaults to 1, the first value.
					Looks up the Value Desc for the *valueName* Value in the Standard Choices called *itemName*. Compares *compareValue* against the series of numbers in the Value Desc. If *valueIndex* is null or 1, calculates the base fee using the value following the 1st pipe (|) on the matching number's right. If *valueIndex* is 2, calculates an add on fee using the value following the 2nd pipe (|) on the matching number's right
					
					Set up the **Standard Choices** lookup table as follows:
					Value column = Lookup value.
					Value Desc column = one or more 3-number series, where
					1st number = number to compare *compareValue* against
					2nd number = base fee
					3rd number = used to calculate add-on fee
					Each number is separated by a pipe(|). Each 3-number series is separated by a caret(^).
					
					<u>Example</u>
					 
					
					`06 true ^ theBase = lookupFeesByValuation("PlanCheck2007","A-1-Group2",5600)`
					`07 true ^ theAddOn = lookupFeesByValuation("PlanCheck2007","A-1-Group2",5600,2)`
					`08 true ^ newTotal = newTotal +(parseFloat(theBase) +parseFloat(theAddOn))`
					
					 

Function	Ver	Category	Type	Parameters	Description
lookupFeesByValuationSlidingScale (stdChoiceEntry, stdChoiceValue, capval, [valueIndex])	1.6	Fee	Get	*stdChoiceEntry (string)* Item Name of Standard Choices used as lookup table *stdChoiceValue (string)* Standard Choices Value *compareValue (number)* Number value (e.g. valuation) to be compared. *valueIndex (integer)(optional)* Determines which value is returned. Defaults to 1, the first value.	Similar to the lookupFeesByValuation function, but introduces another element in the standard choice tables which serves as a divisor for the compareValue. Set up the **Standard Choices** lookup table as follows: **Value** column = Lookup value. **Value Desc** column = one or more 3-number series, where 1st number = number to compare *compareValue* against 2nd number = divisor (e.g., 100, 1000, etc.) 3rd number = base fee 4th number = used to calculate add-on fee Each number is separated by a pipe(|). Each 4-number series is separated by a caret(^).
loopTask (wfTask, wfStatus, wfComment, wfNote, [wfProcess])	1.3	workflow	Edit	*wfTask (string)* workflow task name *wfStatus(string)* status to assign *wfComment (string)* comment to add *wfNote (string)* note to add to the workflow task *wfProcess (optional) (string)* ID (R1_PROCESS_CODE) for the process that the task belongs to. Required for multi-level workflows.	Updates the workflow task *wfTask* as follows: Status = *wfStatus* Status Date = current date Status Comment = *wfComment* Action By = current user `theAddOn` Task *wfTask* is closed and workflow proceeds to the loop task. If application's workflow contains duplicate *wfTask* tasks, use parameter *wfProcess* to specify the process or subprocess whose *wfTask* should be checked.
matches (value, m_1 [, … m_n])	1.3	utility	true/ false	*value (string)* String to match. *m_1 [, …m_n] (strings)* List of values to test for a match. Enter any number of values, each enclosed in double quotes and separated by comma.	Returns **true** if *value* is found in the *m_1 [, … m_n]* list. Function looks for an exact, case-sensitive match. Returns **false** if nothing in the *m_1 [, … m_n]* list matches *value*.
nextWorkDay ([baseDate])	1.4	utility	Get	*baseDate (string)(optional)* Date, in format "MM/DD/YYYY" (or any string that will convert to a JavaScript date).	Returns the first agency work day following the current date, by checking the Agency Workday calendar defined for the agency. If parameter baseDate is used, returns the first agency work day following *baseDate.* The date returned is a string in the format "MM/DD/YYYY". **Note:** Function can only be used with AA 6.3.2 and above.

Function	Ver	Category	Type	Parameters	Description
openUrlInNewWindow (url)	1.4	utility	Utility	*url (string)* *URL of web page to open.*	Opens a new browser window and shows the web page whose URL is *url*. Note: Either user-configurable variable **showDebug** or **showMessage** must be **true** for this function to work.
parcelConditionExists (cType)	1.4	parcel	true/ false	*cType (string)* Condition type.	Returns **true** if any parcel has a condition of type *cType*; otherwise, returns **false**.
parcelExistsOnCap ([capid)	1.6	Record	True/False	*capId (optional)* *capId to check*	Returns true if a parcel exists on the CAP
paymentGetNotAppliedTot ()	1.3	payment	Get	*(none)*	Returns the total amount of non-applied payments on the current CAP, as a float number.
proximity (service, layer, distance, [unit])	1.3	GIS	true/ false	*service (string)* GIS Service name *layer (string)* GIS layer, i.e., object that function is testing proximity to *distance (integer)* Distance of parcel on current app to the GIS object identified by *layer* *unit (string)(optional)* Unit for *distance* measurement. Optional. Default is “feet”.	Returns **true** if the parcel on the current application is within *distance* feet (or other *unit* specified) of the object in *layer;* otherwise, returns nothing.
proximityToAttribute (service, layer, distance, unit, attribute, attributeValue)	1.4	GIS	true/ false	*service (string)* GIS Service name *layer (string)* GIS layer, i.e., object that function is testing proximity to *distance (integer)* Distance of parcel on current app to the GIS object identified by *layer* *unit (string)* Unit for distance measurement. *attribute (string)* Attribute name. attributeValue (string) Attribute value.	Returns **true** if the application has a GIS object in *distance* proximity that contains an attribute called *attribute* with the value *attributeValue.* Example `proximityToAttribute("flagstaff","Parcels","50", "feet","BOOK","107") ^ DoStuff...`

Function	Ver	Category	Type	Parameters	Description
refLicProfGetAttribute (stateLicNum, attributeName)	1.4	Professional	Get	*stateLicNum (string)* State license number. *attributeName (string)* Custom attribute name.	Returns the value of the custom attribute named *attributeName* for the reference Licensed Professional whose license # is *stateLicNum*. Note that *attributeName* is not necessarily the same as the attribute label. The attribute name is found in the attribute's configuration screen. If no reference Licensed Professional with license # of *stateLicNum* is found, the function returns "NO LICENSE FOUND". If the attribute *attributeName* is not found, the function returns "ATTRIBUTE NOT FOUND"
refLicProfGetDate (stateLicNum, dateType)	1.4	Professional	Get	*stateLicNum (string)* State license number. *dateType (string)* Date field to retrieve. Options (use one): EXPIRE, ISSUE, RENEW, INSURANCE, BUSINESS.	Returns the date specified by *dateType* for the reference Licensed Professional whose license # is *stateLicNum*. The table below shows the date returned for each *dateType* parameter value. The date returned is a JavaScript Date object. *dateType* — Date Field Value Returned EXPIRE — License Expiration Date ISSUE — License Issue Date RENEW — License Last Renewal Date INSURANCE — Insurance Expiration Date BUSINESS — Business License Expiration Date If no reference Licensed Professional with license # of *stateLicNum* is found, function returns "NO LICENSE FOUND". If no date is found, function returns "NO DATE FOUND". If stateLicNum is empty, returns "INVALID PARAMETER". Skips disabled reference Licensed Professional. Hint: To format a JavaScript Date as a MM/DD/YYYY string, use function jsDateToMMDDYYYY.
removeAllFees (capId)	1.6	Fee	Edit	*capId (capId)*	Removes all un-invoiced fees on the CAP
removeASITable (tableName, [capId])	1.5	ASI	Edit	*tableName (string)* Table name to remove *capId(CapIDModel) (optional)* CapID object for application	Removes all entries for ASI Table Name
removeCapCondition (cType, cDesc, [capId])	1.5	Condition	Edit	*cType (string)* Condition type. *cDesc (string)* Condition name. *capId (CapIDModel) (optional)* capId object.	Deletes the condition whose type is *cType* and name is *cDesc* from the current CAP. If optional parameter *capId* is used, deletes the condition from the CAP *capId* instead.

Function	Ver	Category	Type	Parameters	Description
removeFee (fcode, fperiod)	1.4	Fee	Edit	*fcode (string)* Fee code of the fee to be deleted. *fperiod (string)* Fee period of the fee to be deleted.	Deletes all assessed fees with the fee code of *fcode* and fee period of *fperiod*. If the fee is invoiced, it is not deleted.
removeParcelCondition (parcelNum, cType, cDesc)	1.4	parcel	Edit	*parcelNum (string)* Parcel number that condition is removed from. If **null** is used, condition will be removed from all parcels on the application. *cType (string)* Condition type. *cDesc (string)* Condition name.	Removes the condition whose name is *cDesc* and type is *cType* from the reference parcel whose number is *parcelNum*. If parameter *parcelNum* is set to **null**, any condition whose name is *cDesc* and type is *cType* will be removed from all parcels on the application.
replaceMessageTokens (messageStr)	1.6	Utility	Utility	*messageStr (string)* *string to do the token replacement*	Used for formatting emails, this function will parse through the string, replacing tokens with variable values. Values surrounded in pipes (e.g. |capIdString|) will be replaced by their script values. Values surrounded in curly brances (e.g. {ASIVal}) will be replaced by ASI values. For example: EmailContent = "Thank you for submitting |capIDString| on |fileDate|. The balance due is |balanceDue|. The ASI field is {ASI Field}" EmailSend = replaceMessageTokens(EmailContent); Any variable used by the script can be accessed by this function.
resultInspection (inspType, inspStatus, resultDate, resultComment, [capId])	1.6	Inspection	Edit	*inspType* *inspection type to result* *inspStatus* *resulting status* *resultDate* *posted date of the result* *resultComment* *comment to add to the result* *capId(optional)* *capId to result*	This function will post a result for a scheduled inspection. If no scheduled inspection exists (of that type for the CAP) then the function will do nothing.

Function	Ver	Category	Type	Parameters	Description
scheduleInspectDate (inspDesc, inspDate, [inspectorID, inspTime, inspComm])	1.5	Inspection	Add	*inspDesc (string)* Inspection type. *inspDate (string)* Scheduled date of inspection. *inspectorID (string) (optional)* User ID of inspector. *inspTime (string) (optional)* Inspection time in HH12:MIAM format or AMPM (e.g. “12:00PM” or “PM”) *inspComm (string) (optional)* Inspection comment.	Schedules the inspection *inspDesc* for the date *inspDate*. If *inspectorID* is supplied, assigns the scheduled inspection to the inspector whose AA user ID is *inspectorID*. Hint To specify the optional inspection time without passing in inspection use `scheduleInspectDate("Desc","01/01/2001",null, "AM").` To specify the option inspection comment without the other option parameters you can use `scheduleInspectDate("Desc","01/01/2001",null,null, "My Comment").`
scheduleInspection (inspDesc, daysAhead, [inspectorID, inspTime, inspComm])	1.5	Inspection	Add	*inspDesc (string)* Inspection type. *daysAhead (number)* Number of days in the future to schedule the inspection for. *inspectorID (string) (optional)* User ID of inspector. *inspTime (string) (optional)* Inspection time in HH12:MIAM format or AMPM (e.g. “12:00PM” or “PM”). *inspComm (string) (optional)* Inspection comment.	Schedules the inspection *inspDesc* for *daysAhead days* after current date. If *inspectorID* is supplied, assigns the scheduled inspection to the inspector whose AA user ID is *inspectorID*. Hint To specify the optional inspection time without passing in inspection use `scheduleInspectDate("Desc",5,null,"AM").` To specify the option inspection comment without the other option parameters you can use `scheduleInspectDate("Desc",5,null,null,"My Comment");`
searchProject (pProjType, pSearchType)	1.6	hierarchy/ related CAPs	Get	*pProjType* (app type string) Application type marking highest point to search. Ex. Building/Project/NA/NA *pSearchType* (app type string) Application type to search for. Ex. Building/Permit/NA/NA	Searches the entire hierarchy on the current CAP for related CAPS that match the criteria. Returns CapID array of all unique matching SearchTypes
setIVR (ivrnum)	1.6	Record	Edit	*Ivrnum (long)* *New IVR tracking number*	Set s the CAP tracking number for IVR
stripNN (fullstr)	1.6	Utility	Utility	*Fullstr (string)* *String to strip*	Strips all non-numeric characters from the string. Only numerals and the period character will remain.
taskCloseAllExcept (wfStatus, wfComment, [$wfTask_1$, … $wfTask_n$])	1.4	workflow	Edit	*wfStatus (string)* Status to assign to tasks. *wfComment (string)* Status comment to add to tasks. *$wfTask_1$ … $wfTask_n$ (string) (optional)* Names of tasks to exclude. Enter one or more tasks separated by commas, each in double-quotes.	Closes all tasks on the application except for tasks in the list *$wfTask_1$… $wfTask_n$*. If only the parameters *wfStatus* and *wfComment* are supplied, all tasks on the application are closed. Before closing each task, this function updates the task as follows: Status = *wfStatus* Status Date = current date Status Comment = *wfComment* Action By = current user

Function	Ver	Category	Type	Parameters	Description
taskStatus (wfTask, [wfProcess, capId])	1.3	workflow	Get	*wfTask (string)* Workflow task name. *wfProcess* (string) (optional) ID (R1_PROCESS_CODE) for the process that the task belongs to. *capId (CapIDModel) (optional)* CapID object for CAP to be used.	Returns the status of the workflow task *wfTask*. If CAP's workflow contains duplicate *wfTask* tasks, use parameter *wfProcess* to specify the process or subprocess whose *wfTask* should be checked. If parameter *capId* is used, function will retrieve data from the CAP *capId*.
taskStatusDate (wfTask, [wfProcess, capId])	1.5	workflow	Get	*wfTask (string)* Workflow task name. *wfProcess* (string) (optional) ID (R1_PROCESS_CODE) for the process that the task belongs to. *capId (CapIDModel) (optional)* CapID object for CAP to be used.	Returns the current status date of the workflow task *wfTask*. If CAP's workflow contains duplicate *wfTask* tasks, use parameter *wfProcess* to specify the process or subprocess whose *wfTask* should be used. If parameter *capId* is used, function will retrieve data from the CAP *capId*.
transferFunds (appNumTo, transferAmt)	1.3	payment	Add	*appNumTo(string)* Application number to transfer funds to. *transferAmt (number: double)* Amount to transfer.	If current application has sufficient funds (i.e. non-applied amount), transfers *transferAmt* from current application to application *appNumTo*. Transaction is recorded as a "Fund Transfer" transaction on both applications. If current application does not have sufficient funds, no fund transfer will take place.
updateAppStatus (status, comment, [capId])	1.3	Record	Edit	*status (string)* Status to update the application to. *comment (string)* Comment to add to status update history. *capId (CapIDModel) (optional)* capId object.	Updates application status of application to *status* and adds *comment* to the status update history. If *capId* optional parameter is used, updates application *capId*. If *capId* parameter is not used, updates current application. Hint: getApplication( ), getParent( ), createChild(), createCap() functions each return a capId object that can be used in the *capId* parameter.

Function	Ver	Category	Type	Parameters	Description
updateFee (fcode, fsched, fperiod, fqty, finvoice, [duplicateFee], [feeSeq])	1.5	Fee	Edit	*fcode (string)* Fee code of the fee to be updated/added. *fsched (string)* Fee schedule of the fee to be updated/added. *fperiod (string)* Fee period of the fee to be updated/added. *fqty (integer)* Quantity to be updated/added. *finvoice (string)* Flag for invoicing (“Y” or “N”). *duplicateFee (string) (optional)* Allow duplicate invoiced fee ("Y" or "N"). *feeSeq (integer) (optional)* *Attempts to update a specific fee item*	If a fee whose fee code is *fcode* and fee period is *fperiod* has been assessed and not invoiced, updates the quantity on the fee to *fqty*. If *finvoice* is "Y", then invoices the fee. If there is more than one assessed fee with *fcode* and *fperiod*, updates the first fee found. If the fee is not found, adds the fee. If this fee already exists and is invoiced, adds another instance of the same fee, unless *duplicateFee* is "N". The duplicate fee has an adjusted quantity, which is *fqty* less quantity on previous fee. If feeSeq is specified it will attempt to find the specified fee, if specified fee sequence number is not found a new fee will be added based upon the duplicateFee fee flag. Function will return null if fee is updated and the fee sequence number if a fee is added. Warning: If adjusted quantity may be negative, do not use this function to add a fee. AA's Cashier feature does not handle negative fees well. Set *duplicateFee* parameter to "N".
updateRefParcelToCap ([capId])	1.6	Record	Edit	*capId(optional)* *capId to process*	Refreshes parcel data on the specified CAP. Parcel data on the CAP will be refreshed with reference parcel values.
updateShortNotes (newSN, [capId])	1.6	Record	Edit	*newSN (string)* *new short notes value* *capId (optional capId)* *capID to update*	Updates the short notes on the specific capId detail record

Function	Ver	Category	Type	Parameters	Description
updateTask (wfTask, wfStatus, wfComment, wfNote, [wfProcess, capId])	1.3	workflow	Edit	*wfTask (string)* Name of workflow task to update. *wfStatus (string)* Status to update task to. *wfComment (string)* Comment to update status comment to. *wfNote (string)* Note to update task note to. *wfProcess (string) (optional)* Workflow process that *wfTask* belongs to. *capId (CapIDModel) (optional)* capId object.	Updates the workflow task *wfTask* as follows: Status = *wfStatus* Status Date = current date Status Comment = *wfComment* Action By = current user The workflow does not proceed to the next task. If workflow should proceed to the next, branch, or loop task--use closeTask, branchTask or loopTask functions instead. If application's workflow contains duplicate *wfTask* tasks, use parameter *wfProcess* to specify the process or subprocess whose *wfTask* should be checked. If *capId* parameter is used, updates the application *capId*. If *capId* parameter is used, then *wfProcess* parameter must be used by either entering a process (string) or the word **null**.
updateTaskAssignedDate (wfstr, wfassignDate, [processCode])	1.6	Workflow	Edit	*wfstr (string)* Workflow task to be edited. *wfassignDate (string representing date)* New assignment date. *wfProcess (string) (optional)* Process name of workflow for *wfTask*. Case sensitive.	Updated the assigned date of the workflow task *wfTask*. No workflow history record is created. If application's workflow contains duplicate *wfTask* tasks, use parameter *wfProcess* to specify the process or subprocess whose *wfTask* should be activated.
updateTaskDepartment (wfstr, wfDepartment, [processCode])	1.6	Workflow	Edit	*wfstr (string)* Workflow task to be edited. *wfDepartment (string representing department)* New department code *wfProcess (string) (optional)* Process name of workflow for *wfTask*. Case sensitive.	Updated the assigned department for the workflow task *wfTask*. No workflow history record is created. If application's workflow contains duplicate *wfTask* tasks, use parameter *wfProcess* to specify the process or subprocess whose *wfTask* should be activated. Assigned department must be a string with 7 values separated by slashes, such as "ADDEV/DPE/ONLINE/LICENSE/NA/NA/NA"
updateWorkDesc (newDesc, [capId])	1.6	Workflow	Edit	*newDesc (string)* *new work description value* *capId (optional capId)* *capID to update*	Updates the work description on the specific capId detail record
validateGisObjects ()	1.3	GIS	true/ false	*(none)*	Returns **true** if all GIS objects on the current application validate in GIS, or **false** if any GIS object on the current application does not validate in GIS.

Function	Ver	Category	Type	Parameters	Description
workDescGet (capId)	1.4	work descrip-tion	Get	*capIdFrom (CapIDModel)* CapID object for application.	Returns work description for the application whose CapID object is *capIdFrom*. Hint: getApplication(), getParent(), createChild(), createCap() functions each return a CapID object.
zeroPad (num, Count)	1.6	Utility	Utility	*Num (string)* *Number to zero pad* *Count (integer)* *Number of digits required*	This function will return a zero-padded string of the supplied number. The result will be *Count* digits long. For example: zeroPad("5",4) = "0005"

Conventions Used in this List

1. In the **Function** column
   a. Function names are in bold.
   b. Function parameters are listed in parentheses in the order they must be entered. Optional parameters are in square brackets [ ]. Do not use the square brackets when using the function in a script control.

2. In the **Parameters** and **Description** columns
   a. Function parameters are in italics.
   b. Functions are listed in the order they must be entered.
   c. The parameter's data type follows the parameter name in parentheses. If the data type is *string*, the parameter value must be enclosed in double-quotes. If the data type is *integer* or *number*, double-quotes are not required.
   d. When using **null** as a parameter value, do not enclose in quotation marks.
   e. Subscripts 1 and n in parameter names (e.g., $wfTask_1$ … $wfTask_n$) indicate that between one and any number of such parameters may be added, each in double-quotes and separated by commas.
   f. Boolean values are shown as **true** or **false**.

Notes

1. If an older script control uses the function **closeWorkflow**, it must be modified to use the function **closeTask** if upgrading to Master Scripts version 1.3 and above.

Updated by John Schomp, Accela Technical Services, 2/3/2010