Events and Actions Technical Information

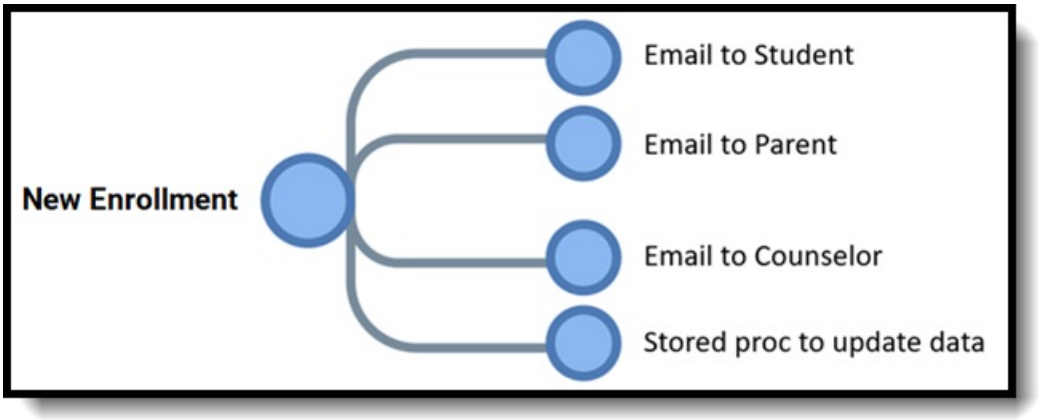
Last Modified on 12/14/2025 8:45 pm CST

What is Events and Actions (EA)?

An Event is a save or delete on a record in the Infinite Campus SIS product. This includes inserts and updates. An Action is an email or execution of a stored procedure. Multiple actions can be executed. Events can have multiple Actions.

Emails can be sent to people in a user group or a query of people with user accounts. A user must have a user account and an email in Infinite Campus.

Stored Procedures (proc) are used to run a query to insert, update, or delete records.



Four areas of Events and Actions can use a stored proc: condition, action, user query, and query.

1. Event - **Conditional** - After selecting the table, column, and potentially the field value change, use a stored proc to further filter the results of the affected data (for example, after a new enrollment is created, find only students that have never been enrolled in the district before).
2. Action - **Action** - Instead of choosing to send an email, a stored proc can be selected to run a query to perform an action (for example, when a new transcript record is added, check if any courses are repeated and, if so, remove the GPA weight of the lower of the two scores so that the lower grade does not negatively affect the student's CUM GPA).
3. Action - **User Query** (recipients) - Users can choose either a user group or stored proc. The stored proc option can be anyone with an Infinite Campus user account and email. i.e. Parent, counselor, case manager, teacher.
4. Action - **Query** - (email fields) - Out of the box, the email body does not contain any ad hoc fields for selection. All data in the email body must be derived from a stored proc (for example, “Hello, your student <studentname> was late to school today <date> at <time of day>.”). It may be useful to have a standard query to pull basic demographic information

that can be used for many emails.

When an event is triggered, the parameters that are passed to each stored proc are the following:

- @IC_EVENT_ACTION_TYPE CHAR(1),
- @IC_EVENT_TIMESTAMP DATETIME,
- @IC_EVENT_TABLE VARCHAR(128),
- @IC_EVENT_COLUMN VARCHAR(128),
- @IC_EVENT_KEY_ID INT,
- @IC_EVENT_PERSON_ID INT,
- @IC_EVENT_CALENDAR_ID INT,
- @IC_EVENT_SCHOOL_ID INT,
- @IC_EVENT_DISTRICT_ID INT,
- @IC_EVENT_DATA VARCHAR(MAX),
- @IC_EVENT_USER_ID INT,
- @IC_EVENT_TOOL_ID VARCHAR(50)

Not all parameters will have values passed. This is dependent on the type of event that occurs and the table that was triggered on. These values can be used to filter the query results of the stored proc.

Stored Proc Parameters and Potential Values

Field	Description
IC_EVENT_ACTION_TYPE	I = Insert, U = Update, D = Delete
IC_EVENT_TIMESTAMP	2020-09-15 00:00:00.000 – The date that the event was triggered on.
IC_EVENT_TABLE	The table that was triggered.
IC_EVENT_COLUMN	The column that was triggered. Values can be NULL.
IC_EVENT_KEY_ID	The primary key value of the table triggered on.
IC_EVENT_PERSON_ID	The personID that the event was triggered on. Values can be NULL.
IC_EVENT_CALENDAR_ID	The calendarID on the record that was triggered. Values can be NULL.
IC_EVENT_SCHOOL_ID	The schoolID on the record that was triggered. Values can be NULL.
IC_EVENT_DISTRICT_ID	The districtID on the record that was triggered. Values can be NULL.
IC_EVENT_DATA	The JSON string of data that was changed.

IC_EVENT_USER_ID	The userID of the person that triggered the event.
IC_EVENT_TOOL_ID	The toolID that was used to trigger the event.

In the example below, an Event was created to trigger when a new fee assignment was created. A conditional stored proc was added to filter out any fee assignment flagged as exempt. These are the values that were passed to the conditional stored proc.

IC_EVENT_ACTION_TYPE: I

IC_EVENT_TIMESTAMP: 2022-03-14 00:00:00.000

IC_EVENT_TABLE: dbo.FeeAssignment

IC_EVENT_COLUMN: NULL

IC_EVENT_KEY_ID: 28163

IC_EVENT_PERSON_ID: 4371

IC_EVENT_CALENDAR_ID: 250

IC_EVENT_SCHOOL_ID: NULL

IC_EVENT_DISTRICT_ID: NULL

IC_EVENT_DATA: {"old": null, "new": { "assignmentID":28163, "calendarID":250, "personID":4371, "feeID":62, "dueDate":"2022-03-16T00:00:00", "amount":50.00, "createdByID":49574, "createdDate":"2022-03-16T13:22:00", "modifiedByID":49574, "modifiedDate":"2022-03-16T13:22:00" }}

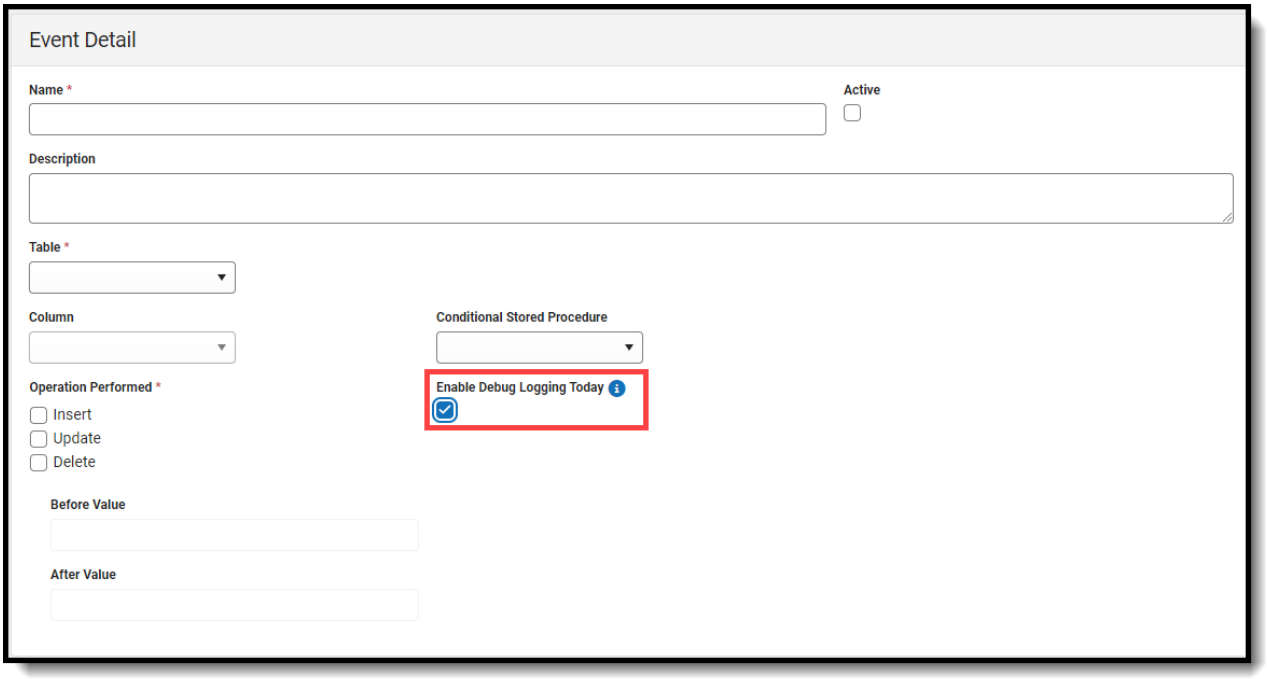
IC_EVENT_USER_ID: 27317

IC_EVENT_TOOL_ID: 696

Below is the conditional stored proc. Notice that the only parameter used to filter on is the @IC_EVENT_KEY_ID which is the feeassignmentID. We can derive the personID of the student that the fee was created for by filtering on this value.

```
CREATE PROCEDURE [cust].[condition_NewFeeAssignment]
@IC_EVENT_ACTION_TYPE CHAR(1),
@IC_EVENT_TIMESTAMP DATETIME,
@IC_EVENT_TABLE VARCHAR(128),
@IC_EVENT_COLUMN VARCHAR(128),
@IC_EVENT_KEY_ID INT,
@IC_EVENT_PERSON_ID INT,
@IC_EVENT_CALENDAR_ID INT,
@IC_EVENT_SCHOOL_ID INT,
@IC_EVENT_DISTRICT_ID INT,
@IC_EVENT_DATA VARCHAR(MAX),
@IC_EVENT_USER_ID INT,
@IC_EVENT_TOOL_ID VARCHAR(50)

AS

SELECT p.personID
FROM person p
INNER JOIN [identity] i ON i.identityID = p.currentidentityID
INNER JOIN enrollment e ON e.personID = p.personID
INNER JOIN feeassignment fa ON fa.personID = p.personID
      AND fa.calendarID = e.calendarID
INNER JOIN fee f ON f.feeID = fa.feeID
WHERE 1=1
AND COALESCE(fa.exempt, 0) = 0
AND fa.assignmentID = @IC_EVENT_KEY_ID
```

It is recommended that when creating new Events and Actions to select the Enable Debug Logging Today on the event. This will allow you to see what happened when the event triggered any status of the actions.

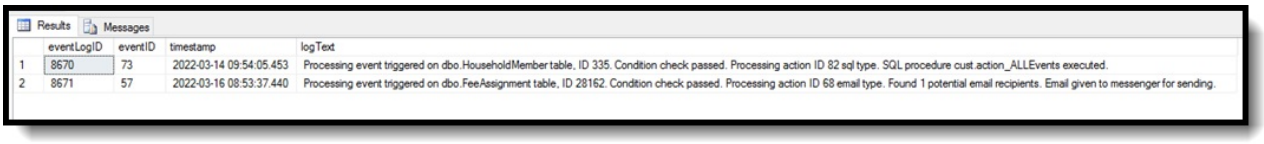


Once an event is triggered, the Campus Event Log table can be queried to review the results of the task.

Enable Debug Logging Today, when checked, will automatically turn off each night. The log will

only keep records in the table for seven days. This was done to keep the size of this table low.

```
SELECT * FROM CampusEventLog ORDER BY 1 DESC
```



	eventLogID	eventID	timestamp	logText
1	8670	73	2022-03-14 09:54:05.453	Processing event triggered on dbo.HouseholdMember table, ID 335. Condition check passed. Processing action ID 82 sql type. SQL procedure cust.action_ALLEvents executed.
2	8671	57	2022-03-16 08:53:37.440	Processing event triggered on dbo.FeeAssignment table, ID 28162. Condition check passed. Processing action ID 68 email type. Found 1 potential email recipients. Email given to messenger for sending.

After the creation of the stored proc in the “cust” schema, the stored proc must be added to the campusEventProcedure table. Adding the procedure to this table will tell Events and Actions where to display the procedure.

procedure = Name of procedure

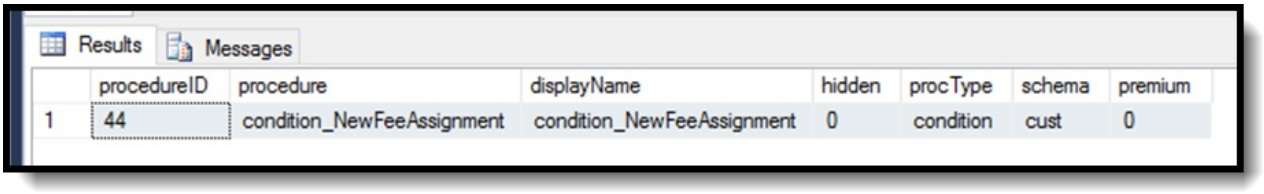
displayName = What the user will see Events and Actions drop lists.

hidden = Always 0

procType – condition, action, userquery, query

Schema -cust

Premuim – Always = 0



	procedureID	procedure	displayName	hidden	procType	schema	premium
1	44	condition_NewFeeAssignment	condition_NewFeeAssignment	0	condition	cust	0

Examples of Events and Actions

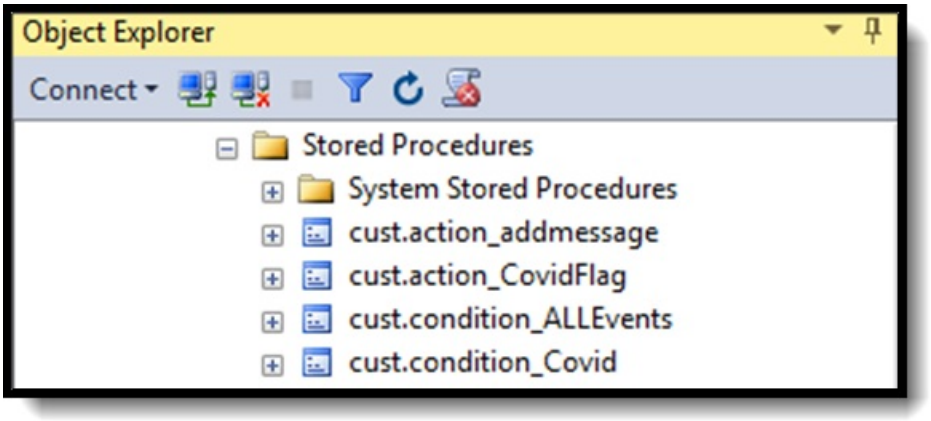
1. Email a student and parent(s) when an assignment is marked as missing.
2. Email a parent when a progress grade fails below the passing score.
3. Email a parent when a student has a daily health screening that would require a student to be placed on quarantine. Then create a “Quarantine flag that is valid for 14 days. An ad hoc filter can be created to only allow students who are not quarantined to enter (scan) into the building using Advanced Attendance and Appointments.
4. When transcript records are posted, run a stored proc to remove GPA weight for all repeated courses of the lower score.

Tips and Tricks

All stored procs should be created with a naming convention to visually distinguish what kind of proc it is, such as condition_NameofProc or action_NameofProc. This will sort each of the procs in alpha order and they can more easily be found when looking in the cust schema in the database. It is recommended that when adding the stored proc to the CampusEventProcedure table you use the same value for the proc name and the display name. There is no reason to have different

values. End users will never see either of these names and using that same value will make it easier to troubleshoot when you are having issues. You may also use your district number or the name of your organization to the proc name as well.

Cust.action_ISD279_NewfeeAssignment



When creating an event, for example on the attendance table, and you want to track inserts, updates, or deletes, make sure that you trigger on a value that changes like excuseID versus attendanceID. If you only select the primary key to trigger on, you will not get a notification due to the primary key not changing when a record is updated.

Create a custom table and stored proc to insert each parameter passed for testing and troubleshooting.

```
Table:
CREATE TABLE [dbo].[eventsActions](
	[ea_ID] [int] IDENTITY(1,1) PRIMARY KEY NOT NULL,
	[IC_EVENT_ACTION_TYPE] [char](1) NULL,
	[IC_EVENT_TIMESTAMP] [datetime] NULL,
	[IC_EVENT_TABLE] [varchar](128) NULL,
	[IC_EVENT_COLUMN] [varchar](128) NULL,
	[IC_EVENT_KEY_ID] [int] NULL,
	[IC_EVENT_PERSON_ID] [int] NULL,
	[IC_EVENT_CALENDAR_ID] [int] NULL,
	[IC_EVENT_SCHOOL_ID] [int] NULL,
	[IC_EVENT_DISTRICT_ID] [int] NULL,
	[IC_EVENT_DATA] [varchar](max) NULL,
	[IC_EVENT_USER_ID] [int] NULL,
	[IC_EVENT_TOOL_ID] [varchar](50) NULL
)

GO
Stored Proc:
CREATE PROCEDURE [cust].[condition_ALLEvents]
@IC_EVENT_ACTION_TYPE CHAR(1),
@IC_EVENT_TIMESTAMP DATETIME,
@IC_EVENT_TABLE VARCHAR(128),
@IC_EVENT_COLUMN VARCHAR(128),
@IC_EVENT_KEY_ID INT,
@IC_EVENT_PERSON_ID INT,
@IC_EVENT_CALENDAR_ID INT,
@IC_EVENT_SCHOOL_ID INT,
@IC_EVENT_DISTRICT_ID INT,
```

```
@IC_EVENT_DATA VARCHAR(MAX),
@IC_EVENT_USER_ID INT,
@IC_EVENT_TOOL_ID VARCHAR(50)

AS

INSERT INTO [dbo].[eventsActions]
([IC_EVENT_ACTION_TYPE]
,[IC_EVENT_TIMESTAMP]
,[IC_EVENT_TABLE]
,[IC_EVENT_COLUMN]
,[IC_EVENT_KEY_ID]
,[IC_EVENT_PERSON_ID]
,[IC_EVENT_CALENDAR_ID]
,[IC_EVENT_SCHOOL_ID]
,[IC_EVENT_DISTRICT_ID]
,[IC_EVENT_DATA]
,[IC_EVENT_USER_ID]
,[IC_EVENT_TOOL_ID])
SELECT
@IC_EVENT_ACTION_TYPE,
@IC_EVENT_TIMESTAMP,
@IC_EVENT_TABLE,
@IC_EVENT_COLUMN,
@IC_EVENT_KEY_ID,
@IC_EVENT_PERSON_ID,
@IC_EVENT_CALENDAR_ID,
@IC_EVENT_SCHOOL_ID,
@IC_EVENT_DISTRICT_ID,
@IC_EVENT_DATA,
@IC_EVENT_USER_ID,
@IC_EVENT_TOOL_ID

SELECT 1

Insert Proc into CampusEventProcedure table:

INSERT INTO CampusEventProcedure ([procedure], displayname, hidden, procType, [schema])
VALUES ('condition_ALLEvents','condition_ALLEvents', 0, 'condition', 'cust')
```

The stored proc will now be available to be selected in the event.

A question that is commonly asked is how to trigger something that has not happened. For example: I would like to send an email reminder to students that an assignment is nearing its due date. Nothing has been triggered to initiate the event, so how could this be accomplished? Another example would be that a district would like to email every principal at the end of the school day a summary/total of things that occurred during that school day like:

- New enrollments
- End dated enrollments
- Behavior events
- Students marked as a full-day absent

Events and Actions is not a scheduled process, but rather a process that is triggered when a user has done something, such as a user clicking save.

One potential way to make this happen is to create a query to locate the data you are looking for

and have this query insert values into a custom table via the Task Scheduler. The task is scheduled to run every day at 4:00 PM. When the task scheduler job is executed, and records are inserted into a custom table, an Event and Action can be created to trigger on when values are inserted into that table and perform an action. So, by using the task scheduler to insert rows into a custom table, we are triggering an event when that insert occurs.

JSON String Parsing Examples

When using EA, the value in using the parameter @IC_EVENT_DATA may not seem inherent. This can be due to not being familiar with JSON strings. After using EA more frequently, there will be a need to test for the new values being inserted or what the values are being updated to. An EA could be triggering on a table insert or update and three separate columns will need to be checked, so the single column selection with the before and after values will not suffice, and a condition stored proc will be needed.

A specific example is an EA that was created to trigger on the identity table looking for students that had the First Language, Home Language, or Language with Friends values inserted or updated to something other than English or American Sign Language. (These questions may vary by state). In the first version, after triggering on an identity insert or change, a condition checks for students that have one of these three values.

```
WHERE 1=1
AND (i.homePrimaryLanguage NOT IN ('EN', 'XA')
OR i.languageAlt NOT IN ('EN', 'XA')
OR i.languageAlt2 NOT IN ('EN', 'XA'))
AND p.personID = @IC_EVENT_PERSON_ID
```

All tests were done changing one of these values, so everything appeared to be working as desired. Once the customer turned this EA on, they immediately received word from someone who stated that a value on the identity was changed, but not one of the values that was being tested for and they received a notification. The issue was that although the student did trigger properly on an identity change, it passed the condition query because the student did have one of these three values previously set, but not by this specific identity change, so the WHERE clause was changed to ONLY look for the values in the JSON/IC_EVENT_DATA string parameter.

```
WHERE 1=1
AND (JSON_VALUE(@IC_EVENT_DATA, '$.new.homePrimaryLanguage') NOT IN ('EN', 'XA')
OR JSON_VALUE(@IC_EVENT_DATA, '$.new.languageAlt') NOT IN ('EN', 'XA')
OR JSON_VALUE(@IC_EVENT_DATA, '$.new.languageAlt2') NOT IN ('EN', 'XA'))
AND p.personID = @IC_EVENT_PERSON_ID
```

By switching to testing the actual data that was changed, the condition stored proc was now filtering out any change made to the identity table that was not one of these three fields.

An Identity event update with a first name change would appear like this. Here is the JSON / IC_EVENT_DATA string parameter.

{"old": { "identityID":52092, "personID":49453, "effectiveDate":"2019-07-21T00:00:00", "lastName":"Milton", "firstName":"Geraldd", "middleName":null, "suffix":null, "alias":null, "gender":"M", "birthdate":null, "ssn":null, "raceEthnicity":"5", "birthCountry":null, "dateEnteredUS":null, "birthVerification":null, "comments":null, "districtID":52, "hispanicEthnicity":"N", "identityGUID":"94B1D24A-D8A9-41B9-967D-B47D8A932994", "lastNamePhonetic":"MILTON", "firstNamePhonetic":"GARALD",

"dateEnteredState":null, "birthCertificate":null, "immigrant":null, "dateEnteredUSSchool":null, "raceEthnicityFed":null, "raceEthnicityDetermination":null, "birthStateNoSIF":null, "birthCity":null, "birthCounty":null, "modifiedByID":49374, "modifiedDate":"2022-03-09T14:41:00", "birthVerificationBIE":null, "refugee":null, "homePrimaryLanguage":null, "stateHispanicEthnicity":null, "birthState":null, "homePrimaryLanguageBIE":null, "homeSecondaryLanguageBIE":null, "languageAlt":null, "languageAlt2":null, "foreignLanguageProficiency":false, "literacyLanguage":false, "legalFirstName":null, "legalLastName":null, "legalMiddleName":null, "legalSuffix":null, "legalGender":null, "usCitizen":null, "visaType":null, "originCountry":null, "hispanicWriteIn":null, "asianWriteIn":null, "caribbeanWriteIn":null, "centralAfricanWriteIn":null, "eastAfricanWriteIn":null, "latinAmericanWriteIn":null, "southAfricanWriteIn":null, "westAfricanWriteIn":null, "blackWriteIn":null, "alaskaNativeWriteIn":null, "americanIndianWriteIn":null, "pacificIslanderWriteIn":null, "easternEuropeanWriteIn":null, "middleEasternWriteIn":null, "northAfricanWriteIn":null, "usEntryType":null, "multipleBirth":false, "languageSurveyDate":null, "certificateOfIndianBlood":false, "birthGender":null, "languageInterpreter":false, "languageAltInterpreter":false, "languageAlt2Interpreter":false }, "new": { "firstName":"Gerald" }}

The above string value is very long and using regular SUBSTRING, RIGHT, LEFT, CHARINDEX functions could be used to pull the specific data you may be looking for. Another easier way to do this is to use JSON value functions. In the above string, the following example could be used to get the value of the raceEthnicity: SELECT JSON_VALUE(@IC_EVENT_DATA, '$.old.raceEthnicity')

An example of creating flags for students that meet some criteria would be when Online Registration (OLR) updates a custom tab with a Media Release opt out, or when a user manually makes this change, that a flag gets created for that student. You would first need to make sure that you are asking this question in OLR and that the data is mapped to the custom tab. You also need to have a flag created. There are two procs listed below to accomplish this. The first is a condition proc that will confirm that the value, with a code of “MRA”, on the custom tab called “OLR Posted Data” is checked. Once this event is triggered, and the condition check passes, the action proc to insert the flag for the student. The flag was named “Media Opt out” with a code of “MRO”. Two variables were declared for the insert. One is finding the programID of the flag needs to be inserted. The other is to set the desired tooltip. This can be done to easily reuse this proc for others flags and only the code, name, and tooltip parameters need to be changed.

```
CREATE PROCEDURE [cust].[condition_MediaFlag]
@IC_EVENT_ACTION_TYPE CHAR(1),
@IC_EVENT_TIMESTAMP DATETIME,
@IC_EVENT_TABLE VARCHAR(128),
@IC_EVENT_COLUMN VARCHAR(128),
@IC_EVENT_KEY_ID INT,
@IC_EVENT_PERSON_ID INT,
@IC_EVENT_CALENDAR_ID INT,
@IC_EVENT_SCHOOL_ID INT,
@IC_EVENT_DISTRICT_ID INT,
@IC_EVENT_DATA VARCHAR(MAX),
@IC_EVENT_USER_ID INT,
@IC_EVENT_TOOL_ID VARCHAR(50)

AS

DECLARE @attributeID As INT
SET @attributeID = (SELECT attributeID FROM CampusAttribute WHERE element = 'MRA' AND [object] = 'OLR Posted Data')

SELECT DISTINCT e.personID
FROM enrollment e
INNER JOIN calendar c ON c.calendarID = e.calendarID
INNER JOIN school s ON s.schoolID = c.schoolID
WHERE 1=1
AND JSON_VALUE(@IC_EVENT_DATA, '$.new.attributeID') = @attributeID
AND e.personID = @IC_EVENT_PERSON_ID
AND JSON_VALUE(@IC_EVENT_DATA, '$.new.value') = '1'
```

```
AND @IC_EVENT_DISTRICT_ID IS NOT NULL

/*
INSERT INTO CampusEventProcedure ([procedure], displayname, hidden, procType, [schema])
VALUES ('condition_MediaFlag','condition_MediaFlag', 0, 'condition', 'cust')
*/

CREATE PROCEDURE [cust].[action_MediaFlag]

@IC_EVENT_ACTION_TYPE CHAR(1),
@IC_EVENT_TIMESTAMP DATETIME,
@IC_EVENT_TABLE VARCHAR(128),
@IC_EVENT_COLUMN VARCHAR(128),
@IC_EVENT_KEY_ID INT,
@IC_EVENT_PERSON_ID INT,
@IC_EVENT_CALENDAR_ID INT,
@IC_EVENT_SCHOOL_ID INT,
@IC_EVENT_DISTRICT_ID INT,
@IC_EVENT_DATA VARCHAR(MAX),
@IC_EVENT_USER_ID INT,
@IC_EVENT_TOOL_ID VARCHAR(50)

AS

DECLARE @programID As INT
SET @programID = (SELECT programID FROM program WHERE code = 'MRO' AND name = 'Media Opt out')

DECLARE @tooltip As VARCHAR(25)
SET @tooltip = 'Media Opt out'

INSERT INTO ProgramParticipation (programID, personID, startdate, enddate,userwarning, districtID, modifiedDate, mo
difiedByID)
SELECT @programID,@IC_EVENT_PERSON_ID, CONVERT(CHAR(10),GETDATE(),101), null,''''+@tooltip+'''', s.districtID,
GETDATE(),@IC_EVENT_USER_ID
FROM enrollment e
INNER JOIN calendar c ON c.calendarID = e.calendarID
INNER JOIN school s ON s.schoolID = c.schoolID
LEFT OUTER JOIN ProgramParticipation pp ON pp.personID = e.personID
      AND pp.programID = @programID
      AND (pp.enddate IS NULL OR pp.enddate >=GETDATE())
WHERE 1=1
AND e.personID = @IC_EVENT_PERSON_ID
AND pp.personID IS NULL

/*
INSERT INTO CampusEventProcedure ([procedure], displayname, hidden, procType, [schema])
VALUES ('action_MediaFlag','action_MediaFlag', 0, 'action', 'cust')
*/
```

NOTE: The issue below has been resolved in a recent update, but is still useful if you are using JSON values in your select statement for the body of the email, adding a string value if the desired value is null.

One thing to be aware of is that any field that you insert into the email body must have a returned value. If a value is null the email will fail to send. So, for any value that you place in your email

body that may potentially be null, wrap this value in a COALESCE or ISNULL function. i.e. COALESCE(e.enddate, 'No End date') AS enddate OR ISNULL(e.enddate, 'No end date') AS enddate.

For example, an EA triggers during an insert, update, and delete. The query for the email parses the old end date and the new end date to display in the email, but the update was just the addition of an end date, not a change to the end date.

```
SELECT i.firstname + ' ' + i.lastname As studentFullName, i.firstname AS studentFirstName, p.studentnumber,
e.grade,CONVERT(CHAR(10),e.startdate,101) AS startdate, c.[name] AS calendarname, s.name AS schoolname, sy.lab
el As schoolYear,
CONVERT(CHAR(10),JSON_VALUE(@IC_EVENT_DATA, '$.old.endDate'),101) As oldEndDate,
CONVERT(CHAR(10),JSON_VALUE(@IC_EVENT_DATA, '$.new.endDate'),101) As NewEndDate,
--CONVERT(CHAR(10),JSON_VALUE(@IC_EVENT_DATA, '$.old.name'),101) As oldRelationship,
--CONVERT(CHAR(10),JSON_VALUE(@IC_EVENT_DATA, '$.new.name'),101) As NewRelationship,
id.firstname + ' ' + id.lastname As modifiedBy, c.name AS CalendarName
FROM person p
INNER JOIN [identity] i ON i.identityID = p.currentIdentityID
INNER JOIN enrollment e ON e.personID = p.personID
INNER JOIN calendar c ON c.calendarID = e.calendarID
INNER JOIN school s ON s.schoolID = c.schoolID
INNER JOIN schoolyear sy ON sy.endyear = c.endyear AND sy.active = 1

LEFT OUTER JOIN useraccount ua ON ua.userID = @IC_EVENT_USER_ID
LEFT OUTER JOIN individual id ON id.personID = ua.personID
WHERE 1=1
AND p.personID = @IC_EVENT_PERSON_ID
```

The email will fail because there is no value for the end date in the JSON string. Adding a COALESCE to the end date field will still produce results.

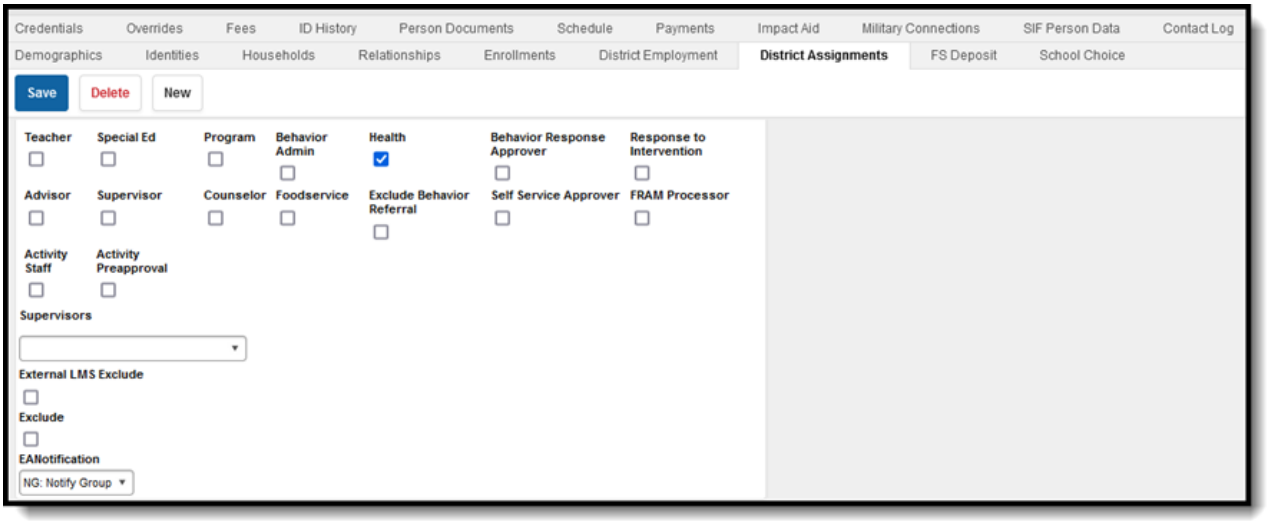
```
COALESCE(CONVERT(CHAR(10),JSON_VALUE(@IC_EVENT_DATA, '$.old.endDate'),101),'No previous End date') As oldEn
dDate,

COALESCE(CONVERT(CHAR(10),JSON_VALUE(@IC_EVENT_DATA, '$.old.endDate'),101),'No previous End date') As oldEn
dDate,
```

Another common scenario is how to send an email to a specific person at a school when something happens, such as sending an email to the Attendance Clerk at one school versus all schools. A best practice is to select “Limit to users with calendar Rights” on the email user group settings. The issue is that many districts give certain users rights to all calendars, so this option does not work for them. One way to dynamically do this is by creating a stored proc for recipients to look at employment assignment versus calendar rights within user groups.

Below is an example that uses a stored proc to identify specific people at the school where the event occurred. This query is linked to the employment assignment table and looks for a specific employment title “Attendance Clerk”. This could easily be changed to any title or another field on employment assignment like department, assignment code, supervisor, etc.

```
CREATE PROCEDURE [cust].[userquery_schoolAssignment]
@IC_EVENT_ACTION_TYPE CHAR(1),
@IC_EVENT_TIMESTAMP DATETIME,
@IC_EVENT_TABLE VARCHAR(128),
@IC_EVENT_COLUMN VARCHAR(128),
@IC_EVENT_KEY_ID INT,
@IC_EVENT_PERSON_ID INT,
@IC_EVENT_CALENDAR_ID INT,
@IC_EVENT_SCHOOL_ID INT,
@IC_EVENT_DISTRICT_ID INT,
@IC_EVENT_DATA VARCHAR(MAX),
@IC_EVENT_USER_ID INT,
@IC_EVENT_TOOL_ID VARCHAR(50)

AS

SELECT DISTINCT ua.userID
FROM person p
INNER JOIN [identity] i ON i.identityID = p.currentIdentityID
INNER JOIN enrollment e ON e.personID = p.personID
INNER JOIN calendar c ON c.calendarID = e.calendarID
INNER JOIN school s ON s.schoolID = c.schoolID

INNER JOIN EmploymentAssignment ea ON ea.schoolID = c.schoolID
      AND (ea.startdate <= GETDATE() AND (ea.enddate IS NULL OR ea.enddate >= GETDATE()))
INNER JOIN person p2 ON p2.personID = ea.personID
INNER JOIN [identity] i2 ON i2.identityID = p2.currentIdentityID
INNER JOIN useraccount ua ON ua.personID = p2.personID
WHERE 1=1
AND p.personID = @IC_EVENT_PERSON_ID
AND e.calendarID = @IC_EVENT_CALENDAR_ID
AND e.enrollmentID = @IC_EVENT_KEY_ID
AND ea.title = 'Attendance Clerk'

/*
INSERT INTO CampusEventProcedure ([procedure], displayname, hidden, procType, [schema])
VALUES ('userquery_schoolAssignment','userquery_schoolAssignment', 0, 'userquery', 'cust')

*/

GO
```

The next example is using a custom attribute drop list on the employment assignment. The EANotifications drop-down list attribute can be used to identify types of notifications.




```
CREATE PROCEDURE [cust].[userquery_stuNumPopulated]

@IC_EVENT_ACTION_TYPE CHAR(1),
@IC_EVENT_TIMESTAMP DATETIME,
@IC_EVENT_TABLE VARCHAR(128),
@IC_EVENT_COLUMN VARCHAR(128),
@IC_EVENT_KEY_ID INT,
@IC_EVENT_PERSON_ID INT,
@IC_EVENT_CALENDAR_ID INT,
@IC_EVENT_SCHOOL_ID INT,
@IC_EVENT_DISTRICT_ID INT,
@IC_EVENT_DATA VARCHAR(MAX),
@IC_EVENT_USER_ID INT,
@IC_EVENT_TOOL_ID VARCHAR(50)

AS

SELECT DISTINCT ua.userID--, s.schoolID, i2.lastname, i2.firstname
FROM person p
INNER JOIN [identity] i ON i.identityID = p.currentIdentityID
INNER JOIN enrollment e ON e.personID = p.personID
INNER JOIN calendar c ON c.calendarID = e.calendarID
INNER JOIN school s ON s.schoolID = c.schoolID
INNER JOIN schoolyear sy ON sy.endyear = c.endyear AND sy.active = 1
INNER JOIN EmploymentAssignment ea ON ea.schoolID = c.schoolID
      AND (ea.startdate <= GETDATE() AND (ea.enddate IS NULL OR ea.enddate >= GETDATE()))
INNER JOIN campusattribute ca ON ca.[object] = 'EmploymentAssignment' AND ca.element = 'EANotify'
INNER JOIN CustomEmploymentAssignment cea ON cea.attributeID = ca.attributeID AND cea.assignmentID = ea.assig
nmentID AND cea.[value] = 'NG'
INNER JOIN person p2 ON p2.personID = ea.personID
INNER JOIN [identity] i2 ON i2.identityID = p2.currentIdentityID
INNER JOIN useraccount ua ON ua.personID = p2.personID
WHERE 1=1
AND p.personID = @IC_EVENT_PERSON_ID
AND (e.startdate < = GETDATE() AND (e.enddate IS NULL OR e.enddate >=GETDATE()))

/*
INSERT INTO CampusEventProcedure ([procedure], displayname, hidden, procType, [schema])
VALUES ('userquery_stuNumPopulatedt','userquery_stuNumPopulated', 0, 'userquery', 'cust')

*/
```