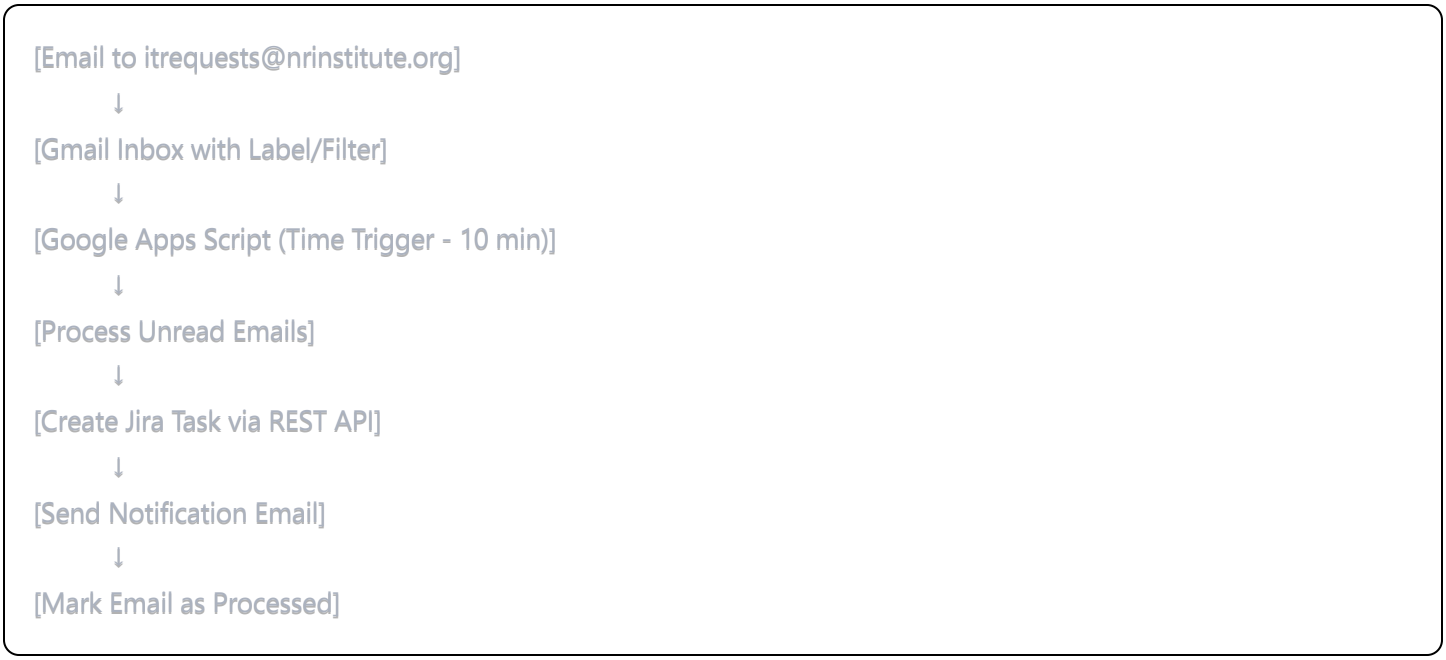
Jira Email Automation Implementation Guide

Architecture Overview



Prerequisites

1. Jira Setup Requirements

- **Jira Cloud Instance URL** (e.g., `https://yourcompany.atlassian.net`)
- **Jira API Token** (Generate from: https://id.atlassian.com/manage-profile/security/api-tokens)
- **Jira Project Key** (e.g., "IT", "HELP")
- **Jira Issue Type** (typically "Task" or "Service Request")
- **Your Jira Email** (for authentication)

2. Google Workspace Requirements

- Admin access to Google Workspace
- Access to itrequests@nrinstitute.org mailbox
- Google Apps Script enabled

3. Gmail Setup

Create a label in the itrequests@nrinstitute.org inbox:

- Label name: "Processed" or "Jira-Created"
- This will track which emails have been processed

Step-by-Step Implementation

Step 1: Create the Google Apps Script

1. Go to https://script.google.com
2. Create a new project
3. Name it "Jira Email Automation"
4. Copy the complete code below

Step 2: Complete Code Implementation

javascript

```
// Configuration - Update these values
const CONFIG = {
  jira: {
    baseUrl: 'https://YOUR-COMPANY.atlassian.net', // Replace with your Jira URL
    email: 'your-email@company.com', // Your Jira account email
    apiToken: '', // Will be set in Script Properties
    projectKey: 'IT', // Your Jira project key
    issueType: 'Task' // or 'Service Request', 'Bug', etc.
  },
  email: {
    notificationRecipient: 'bwilson@nationalreview.com',
    processedLabel: 'Jira-Created', // Label to mark processed emails
    maxEmailsPerRun: 10 // Process maximum 10 emails per run
  }
};

/**
 * Main function to process emails and create Jira tasks
 */
function processEmailsToJira() {
  try {
    console.log('Starting email processing...');

    // Get unprocessed emails
    const emails = getUnprocessedEmails();

    if (emails.length === 0) {
      console.log('No new emails to process');
      return;
    }

    console.log(`Found ${emails.length} emails to process`);

    // Process each email
    emails.forEach(email => {
      try {
        // Create Jira task
        const jiraTask = createJiraTask(email);

        // Send notification email
        sendNotificationEmail(email, jiraTask);

        // Mark email as processed
```

```
        markEmailAsProcessed(email);

        console.log(`Successfully processed email: ${email.subject}`);
      } catch (error) {
        console.error(`Error processing email "${email.subject}":`, error);
        // Continue processing other emails even if one fails
      }
    });

  } catch (error) {
    console.error('Fatal error in processEmailsToJira:', error);
    throw error;
  }
}

/**
 * Get unprocessed emails from inbox
 */
function getUnprocessedEmails() {
  const processedLabel = getOrCreateLabel(CONFIG.email.processedLabel);

  // Search for emails not labeled as processed
  const query = `in:inbox -label:${CONFIG.email.processedLabel}`;
  const threads = GmailApp.search(query, 0, CONFIG.email.maxEmailsPerRun);

  const emails = [];

  threads.forEach(thread => {
    const messages = thread.getMessages();
    // Get the first message in the thread (original email)
    const message = messages[0];

    emails.push({
      id: message.getId(),
      subject: message.getSubject(),
      body: message.getPlainBody(),
      htmlBody: message.getBody(),
      from: message.getFrom(),
      date: message.getDate(),
      thread: thread
    });
  });

  return emails;
```

```
}

/**
 * Create Jira task from email
 */
function createJiraTask(email) {
  const apiToken = PropertiesService.getScriptProperties().getProperty('JIRA_API_TOKEN');

  if (!apiToken) {
    throw new Error('Jira API token not configured. Please set it in Script Properties.');
  }

  // Prepare Jira issue data
  const issueData = {
    fields: {
      project: {
        key: CONFIG.jira.projectKey
      },
      summary: email.subject || 'No Subject',
      description: formatJiraDescription(email),
      issuetype: {
        name: CONFIG.jira.issueType
      },
      reporter: {
        emailAddress: CONFIG.jira.email
      }
      // Not assigning to anyone as per requirements
    }
  };

  // Make API call to create Jira issue
  const response = UrlFetchApp.fetch(`${CONFIG.jira.baseUrl}/rest/api/3/issue`, {
    method: 'POST',
    headers: {
      'Authorization': 'Basic ' + Utilities.base64Encode(`${CONFIG.jira.email}:${apiToken}`),
      'Accept': 'application/json',
      'Content-Type': 'application/json'
    },
    payload: JSON.stringify(issueData),
    muteHttpExceptions: true
  });

  if (response.getResponseCode() !== 201) {
    const error = response.getContentText();
```

```
    throw new Error(`Failed to create Jira task: ${error}`);
  }

  const jiraResponse = JSON.parse(response.getContentText());

  return {
    id: jiraResponse.id,
    key: jiraResponse.key,
    url: `${CONFIG.jira.baseUrl}/browse/${jiraResponse.key}`
  };
}

/**
 * Format email content for Jira description
 */
function formatJiraDescription(email) {
  const description = {
    type: 'doc',
    version: 1,
    content: [
      {
        type: 'paragraph',
        content: [
          {
            type: 'text',
            text: `From: ${email.from}`,
            marks: [{ type: 'strong' }]
          }
        ]
      },
      {
        type: 'paragraph',
        content: [
          {
            type: 'text',
            text: `Date: ${email.date}`,
            marks: [{ type: 'strong' }]
          }
        ]
      },
      {
        type: 'rule'
      },
      {
```

```
      type: 'paragraph',
      content: [
        {
          type: 'text',
          text: 'Email Content:'
        }
      ]
    },
    {
      type: 'paragraph',
      content: [
        {
          type: 'text',
          text: email.body || 'No content'
        }
      ]
    }
  ]
 };

 return description;
}

/**
 * Send notification email about created Jira task
 */
function sendNotificationEmail(email, jiraTask) {
 const subject = `New Jira Task Created: ${jiraTask.key}`;

 const htmlBody = `
   <div style="font-family: Arial, sans-serif; max-width: 600px;">
     <h2 style="color: #0052CC;">New Jira Task Created</h2>

     <div style="background-color: #f4f5f7; padding: 15px; border-radius: 5px; margin: 20px 0;">
       <p><strong>Task ID:</strong> ${jiraTask.key}</p>
       <p><strong>Original Email Subject:</strong> ${email.subject}</p>
       <p><strong>From:</strong> ${email.from}</p>
       <p><strong>Received:</strong> ${email.date}</p>
     </div>

     <div style="margin: 20px 0;">
       <a href="${jiraTask.url}"
          style="background-color: #0052CC; color: white; padding: 10px 20px;
                 text-decoration: none; border-radius: 5px; display: inline-block;">
```

```
        View Task in Jira
      </a>
    </div>

    <hr style="border: none; border-top: 1px solid #ddd; margin: 30px 0;">

    <p style="color: #666; font-size: 12px;">
      This is an automated notification from the IT Request Email Processing System.
    </p>
  </div>
`;

  const plainBody = `
New Jira Task Created

Task ID: ${jiraTask.key}
Original Email Subject: ${email.subject}
From: ${email.from}
Received: ${email.date}

View Task: ${jiraTask.url}

This is an automated notification from the IT Request Email Processing System.
`;

  GmailApp.sendEmail(
    CONFIG.email.notificationRecipient,
    subject,
    plainBody,
    {
      htmlBody: htmlBody,
      name: 'Jira Automation System'
    }
  );
}

/**
 * Mark email as processed
 */
function markEmailAsProcessed(email) {
  const label = getOrCreateLabel(CONFIG.email.processedLabel);
  email.thread.addLabel(label);

  // Optional: Mark as read
```

```
  email.thread.markRead();
}

/**
 * Get or create Gmail label
 */
function getOrCreateLabel(labelName) {
  let label = GmailApp.getUserLabelByName(labelName);

  if (!label) {
    label = GmailApp.createLabel(labelName);
  }

  return label;
}

/**
 * Setup function - Run this once to configure the script
 */
function setup() {
  // Create time-based trigger
  ScriptApp.newTrigger('processEmailsToJira')
    .timeBased()
    .everyMinutes(10)
    .create();

  console.log('Trigger created: Will run every 10 minutes');

  // Create the processed label
  getOrCreateLabel(CONFIG.email.processedLabel);
  console.log(`Label "${CONFIG.email.processedLabel}" created or verified`);

  // Test Jira connection
  testJiraConnection();
}

/**
 * Test Jira connection and permissions
 */
function testJiraConnection() {
  const apiToken = PropertiesService.getScriptProperties().getProperty('JIRA_API_TOKEN');

  if (!apiToken) {
    console.error('❌ Jira API token not set. Please configure it in Script Properties.');
```

```
    return false;
  }

  try {
    const response = UrlFetchApp.fetch(`${CONFIG.jira.baseUrl}/rest/api/3/myself`, {
      method: 'GET',
      headers: {
        'Authorization': 'Basic ' + Utilities.base64Encode(`${CONFIG.jira.email}:${apiToken}`),
        'Accept': 'application/json'
      },
      muteHttpExceptions: true
    });

    if (response.getResponseCode() === 200) {
      const user = JSON.parse(response.getContentText());
      console.log(`✅ Jira connection successful! Connected as: ${user.displayName}`);
      return true;
    } else {
      console.error(`❌ Jira connection failed: ${response.getContentText()}`);
      return false;
    }
  } catch (error) {
    console.error('❌ Error testing Jira connection:', error);
    return false;
  }
}

/**
 * Manual test function
 */
function testWithSampleEmail() {
  // Create a test email object
  const testEmail = {
    subject: 'TEST: Printer not working in Conference Room B',
    body: 'The printer in Conference Room B is showing an error message. Please help!',
    from: 'test@example.com',
    date: new Date(),
    id: 'test-' + new Date().getTime()
  };

  try {
    console.log('Creating test Jira task...');
    const jiraTask = createJiraTask(testEmail);
    console.log('✅ Test task created:', jiraTask);
```

```
    console.log('Sending test notification...');
    sendNotificationEmail(testEmail, jiraTask);
    console.log('✅ Test notification sent');

  } catch (error) {
    console.error('❌ Test failed:', error);
  }
}

/**
 * Remove all triggers (useful for cleanup)
 */
function removeAllTriggers() {
  const triggers = ScriptApp.getProjectTriggers();
  triggers.forEach(trigger => {
    ScriptApp.deleteTrigger(trigger);
  });
  console.log(`Removed ${triggers.length} trigger(s)`);
}
```

Step 3: Configure Script Properties

1. In the Apps Script editor, click on the gear icon (Project Settings)
2. Scroll down to "Script Properties"
3. Add a property:
   - Property: `JIRA_API_TOKEN`
   - Value: Your Jira API token

Step 4: Update Configuration

In the CONFIG object at the top of the script, update:

- `baseUrl`: Your Jira instance URL
- `email`: Your Jira account email
- `projectKey`: Your Jira project key
- `issueType`: The type of issue to create

Step 5: Initial Setup and Testing

1. Run the `setup()` function once to:
   - Create the time-based trigger

- Create the Gmail label
- Test Jira connection

2. Run `testWithSampleEmail()` to:
   - Create a test Jira task
   - Send a test notification email
   - Verify everything works

Step 6: Deploy in Production

1. Save the script
2. Run the `setup()` function
3. Grant necessary permissions when prompted
4. The script will now run every 10 minutes automatically

Testing Checklist

- [ ] Jira API token is configured in Script Properties
- [ ] CONFIG values are updated with correct information
- [ ] `testJiraConnection()` returns success
- [ ] `testWithSampleEmail()` creates a task successfully
- [ ] Notification email is received at bwilson@nationalreview.com
- [ ] Gmail label "Jira-Created" is created
- [ ] Time trigger is set up (check under Triggers in Apps Script)
- [ ] Send a test email to itrequests@nrinstitute.org
- [ ] Verify Jira task is created within 10 minutes
- [ ] Verify notification email contains working Jira link

Troubleshooting

Common Issues and Solutions

1. **"Jira API token not configured"**
   - Add the API token in Script Properties
   - Generate token at: https://id.atlassian.com/manage-profile/security/api-tokens
2. **"Failed to create Jira task: 401 Unauthorized"**
   - Verify email and API token are correct
   - Ensure the email has permissions in Jira

3. **"Project not found"**
   - Verify the project key exists in Jira
   - Check user has access to the project
4. **Emails not being processed**
   - Check the Gmail label is created
   - Verify the trigger is running (check Executions in Apps Script)
   - Look at the execution logs for errors
5. **Notification emails not sending**
   - Verify recipient email is correct
   - Check Gmail sending quotas haven't been exceeded

Security Considerations

1. **API Token Storage**: Store in Script Properties, never hardcode
2. **Access Control**: Limit script access to necessary personnel
3. **Email Filtering**: Consider adding spam/security filters
4. **Rate Limiting**: Script processes max 10 emails per run to avoid quotas
5. **Error Handling**: Errors are logged but don't stop processing

Monitoring and Maintenance

Regular Checks

- Review execution logs weekly
- Monitor failed executions
- Check Jira for duplicate tasks
- Verify notification emails are being received

Logs Location

- Apps Script Editor → Executions tab
- View logs for each run
- Set up email alerts for failures (optional)

Advanced Features (Optional Enhancements)

1. Priority Detection

Add logic to detect keywords and set Jira priority:

```javascript
function detectPriority(emailContent) {
  const urgent = /urgent|asap|critical|emergency/i;
  if (urgent.test(emailContent)) return 'High';
  return 'Medium';
}
```

2. Auto-Assignment Rules

Assign based on email content or sender:

```javascript
function determineAssignee(email) {
  if (email.subject.includes('network')) return 'network-team';
  if (email.subject.includes('printer')) return 'helpdesk-team';
  return null; // Unassigned
}
```

3. Custom Fields

Add department, location, or other custom fields:

```javascript
customfield_10001: 'IT Request',
customfield_10002: extractLocation(email.body)
```

Next Steps with Brian

When ready to test deployment:

1. Show Brian this documentation
2. Request access to:
   - Google Workspace Admin console
   - itrequests@nrinstitute.org mailbox

- Jira project configuration

3. Schedule deployment during low-traffic period
4. Plan monitoring for first 24-48 hours

Support Resources

- Jira REST API Documentation
- Google Apps Script Gmail Service
- Apps Script Triggers Guide
- Jira API Token Management