Bi-Directional Sync with Conflict Resolution

Overview

Bi-directional sync means changes flow both ways: Salesforce → Genesys AND Genesys → Salesforce. This is more complex than one-way sync because conflicts can occur when both systems are updated simultaneously.

Conflict Example:

```
Same contact updated at same time:
  Salesforce: phone changed to +15551111111 at 10:00:05
  Genesys: phone changed to +15552222222 at 10:00:06

Question: Which phone number wins?
```

Conflict Resolution Strategies

Strategy 1: Last-Write-Wins (Simplest)

The system modified MOST RECENTLY wins:

```
function resolveConflict(sfContact, gzContact) {
  const sfTime = new Date(sfContact.LastModifiedDate).getTime();
  const gzTime = new Date(gzContact.dateModified).getTime();

  if (sfTime > gzTime) {
    console.log(`Conflict: SF wins (${sfTime} > ${gzTime})`);
    return { winner: 'salesforce', action: 'update_genesys' };
```

```
  } else {
    console.log(`Conflict: Genesys wins (${gzTime} >= ${sfTime})`);
    return { winner: 'genesys', action: 'update_salesforce' };
  }
}
```

Pros:

- Simple
- No manual intervention
- Works for most cases

Cons:

- Unpredictable (users don't know who wins)
- Possible data loss
- No audit trail

Strategy 2: Field-Level Priority

Different fields have different masters:

```
const fieldPriority = {
  // Names are always SF (master data)
  'firstName': 'salesforce',
  'lastName': 'salesforce',

  // Phone can be updated in either (last-write-wins)
  'phoneNumbers': 'last_write',
  'email': 'last_write',

  // Tags are Genesys (agent annotations)
  'tags': 'genesys',
  'notes': 'genesys',

  // Organization always SF
  'organization': 'salesforce'
};

function applyFieldLevelResolution(sfContact, gzContact) {
```

```
  const merged = {};

  for (const field in fieldPriority) {
    const priority = fieldPriority[field];

    if (priority === 'salesforce') {
      // Always use SF value
      merged[field] = sfContact[field];
    } else if (priority === 'genesys') {
      // Always use Genesys value
      merged[field] = gzContact[field];
    } else if (priority === 'last_write') {
      // Use whoever was modified last
      const sfTime = sfContact.LastModifiedDate || 0;
      const gzTime = gzContact.dateModified || 0;
      merged[field] = new Date(sfTime) > new Date(gzTime)
        ? sfContact[field]
        : gzContact[field];
    }
  }

  return merged;
}
```

Pros:

- Logical (names stay in SF, notes stay in GZ)
- Flexible
- Reduces conflicts

Cons:

- More complex
- Requires configuration
- Still possible data loss

Strategy 3: Timestamp-Based with Locking

Most robust: lock during sync, use timestamps, log everything:

```
async function syncWithLocking(sfContact, gzContact) {
  // 1. Lock both records (prevent other changes during sync)
  await lockSalesforceRecord(sfContact.Id);
  await lockGenesysRecord(gzContact.id);

  try {
    // 2. Detect if either was modified since we last synced
    const lastSyncTime = new Date(sfContact.Last_Sync__c);
    const sfModified = new Date(sfContact.LastModifiedDate) > lastSyncTime;
    const gzModified = new Date(gzContact.dateModified) > lastSyncTime;

    if (!sfModified && !gzModified) {
      // No changes, nothing to do
      return { status: 'no_change' };
    }

    if (sfModified && !gzModified) {
      // Only SF changed, update Genesys
      await updateGenesysContact(gzContact.id, sfContact);
      return { status: 'sf_updated_gz' };
    }

    if (!sfModified && gzModified) {
      // Only Genesys changed, update SF
      await updateSalesforceContact(sfContact.Id, gzContact);
      return { status: 'gz_updated_sf' };
    }

    // CONFLICT: Both changed
    const resolution = await resolveConflictWithLogging(sfContact, gzContact);

    if (resolution.winner === 'salesforce') {
      await updateGenesysContact(gzContact.id, sfContact);
    } else {
      await updateSalesforceContact(sfContact.Id, gzContact);
    }

    // 3. Log conflict for audit
```

```
    await logConflict({
      timestamp: new Date(),
      contactId: sfContact.Id,
      sfLastModified: sfContact.LastModifiedDate,
      gzLastModified: gzContact.dateModified,
      winner: resolution.winner,
      differences: resolution.differences,
      action: resolution.action
    });

    return { status: 'conflict_resolved', winner: resolution.winner };

  } finally {
    // 4. Unlock records
    await unlockSalesforceRecord(sfContact.Id);
    await unlockGenesysRecord(gzContact.id);
  }
}
```

Pros:

- Most reliable
- Prevents race conditions
- Complete audit trail
- Can alert on conflicts

Cons:

- Complex
- Slower (locking overhead)
- Requires timestamp tracking

Implementation: Full Bi-Directional Sync

```
class BidirectionalSync {
  constructor(config) {
    this.config = config;
```

```
  this.stats = {
    synced: 0,
    conflicts: 0,
    errors: 0
  };
  this.conflicts = [];
}

/**
 * Main sync method
 */
async runSync() {
  console.log('\n=== BIDIRECTIONAL SYNC START ===');

  try {
    // 1. Fetch from both systems
    const sfContacts = await this.fetchAllSalesforceContacts();
    const gzContacts = await this.fetchAllGenesysContacts();

    console.log(`Salesforce: ${sfContacts.length} contacts`);
    console.log(`Genesys: ${gzContacts.length} contacts`);

    // 2. Index for matching
    const sfById = new Map(sfContacts.map(c => [c.Id, c]));
    const sfByEmail = new Map(sfContacts.map(c => [c.Email?.toLowerCase(), c]));
    const gzByExtId = new Map(gzContacts.map(c => [c.externalId, c]));

    // 3. Find and sync all contact pairs
    const synced = new Set();

    // Process Salesforce contacts
    for (const sf of sfContacts) {
      let gz = gzByExtId.get(sf.Id) // Match by externalId first
            || sfByEmail.get(sf.Email?.toLowerCase()); // Then by email

      if (gz) {
        // Both systems have it
        await this.syncPair(sf, gz);
        synced.add(sf.Id);
      } else {
```

```
        // Only in Salesforce, create in Genesys
        await this.createInGenesys(sf);
        synced.add(sf.Id);
      }
    }

    // Process Genesys-only contacts
    for (const gz of gzContacts) {
      if (!synced.has(gz.externalId)) {
        // Only in Genesys, create in Salesforce
        await this.createInSalesforce(gz);
      }
    }

    this.printResults();

  } catch (error) {
    console.error('❌ SYNC FAILED:', error);
    throw error;
  }
}

/**
 * Sync a contact that exists in both systems
 */
async syncPair(sfContact, gzContact) {
  try {
    // Detect changes
    const sfModified = this.isModifiedSinceLast(sfContact);
    const gzModified = this.isModifiedSinceLast(gzContact);

    if (!sfModified && !gzModified) {
      // No changes
      return;
    }

    if (sfModified && !gzModified) {
      // SF changed, update Genesys
      await this.updateGenesys(gzContact.id, sfContact);
      this.stats.synced++;
```

```
      return;
    }

    if (!sfModified && gzModified) {
      // Genesys changed, update SF
      await this.updateSalesforce(sfContact.Id, gzContact);
      this.stats.synced++;
      return;
    }

    // CONFLICT: Both changed
    await this.handleConflict(sfContact, gzContact);

  } catch (error) {
    console.error(`Error syncing ${sfContact.Id}:`, error);
    this.stats.errors++;
  }
}

/**
 * Handle conflict between SF and Genesys
 */
async handleConflict(sfContact, gzContact) {
  const resolution = this.resolveConflict(sfContact, gzContact);

  console.warn(`⚠ CONFLICT for ${sfContact.Email}:`);
  console.warn(`   SF: ${sfContact.LastModifiedDate}`);
  console.warn(`   GZ: ${gzContact.dateModified}`);
  console.warn(`   Winner: ${resolution.winner}`);

  // Apply resolution
  if (resolution.winner === 'salesforce') {
    await this.updateGenesys(gzContact.id, sfContact);
  } else {
    await this.updateSalesforce(sfContact.Id, gzContact);
  }

  // Log for review
  this.conflicts.push({
    email: sfContact.Email,
```

```
    sfTime: sfContact.LastModifiedDate,
    gzTime: gzContact.dateModified,
    winner: resolution.winner,
    changes: resolution.changes
  });

  this.stats.conflicts++;
  this.stats.synced++;
}

/**
 * Resolve conflict (strategy: last-write-wins with field priority)
 */
resolveConflict(sfContact, gzContact) {
  const fieldPriority = {
    firstName: 'sf',
    lastName: 'sf',
    email: 'last_write',
    phoneNumbers: 'last_write',
    externalOrganization: 'sf',
    tags: 'gz',
    attributes: 'last_write'
  };

  const sfTime = new Date(sfContact.LastModifiedDate).getTime();
  const gzTime = new Date(gzContact.dateModified).getTime();
  const overallWinner = sfTime > gzTime ? 'salesforce' : 'genesys';

  const merged = {};
  const changes = {};

  for (const field in fieldPriority) {
    const priority = fieldPriority[field];
    let value;

    if (priority === 'sf') {
      value = sfContact[field];
    } else if (priority === 'gz') {
      value = gzContact[field];
    } else {
```

```
      // last_write
      value = sfTime > gzTime ? sfContact[field] : gzContact[field];
    }

    if (sfContact[field] !== gzContact[field]) {
      changes[field] = {
        sf: sfContact[field],
        gz: gzContact[field],
        winner: priority === 'sf' ? 'sf' : priority === 'gz' ? 'gz' : overallWinner
      };
    }

    merged[field] = value;
  }

  return {
    winner: overallWinner,
    merged,
    changes
  };
}

/**
 * Check if contact was modified since last sync
 */
isModifiedSinceLast(contact) {
  if (contact.Last_Sync__c) {
    const lastSync = new Date(contact.Last_Sync__c).getTime();
    const lastMod = new Date(contact.LastModifiedDate || contact.dateModified).getTime();
    return lastMod > lastSync;
  }
  return true; // Assume modified if no last sync
}

/**
 * Update Genesys with SF data
 */
async updateGenesys(gzId, sfContact) {
  const updateData = {
    firstName: sfContact.FirstName,
```

```
    lastName: sfContact.LastName,
    email: sfContact.Email,
    phoneNumbers: sfContact.Phone ?
      [{ number: sfContact.Phone, type: 'WORK' }] : [],
    externalId: sfContact.Id  // Keep the link
  };

  await axios.patch(
    `https://api.mypurecloud.com/api/v2/contacts/${gzId}`,
    updateData,
    { headers: { 'Authorization': `Bearer ${this.gzToken}` } }
  );

  console.log(`  Updated Genesys: ${gzId}`);
}

/**
 * Update Salesforce with Genesys data
 */
async updateSalesforce(sfId, gzContact) {
  const updateData = {
    FirstName: gzContact.firstName,
    LastName: gzContact.lastName,
    Email: gzContact.email,
    Phone: gzContact.phoneNumbers?.[0]?.number,
    Last_Sync__c: new Date().toISOString()
  };

  await axios.patch(
    `${process.env.SALESFORCE_INSTANCE}/services/data/v57.0/sobjects/Contact/${sfId}`,
    updateData,
    { headers: { 'Authorization': `Bearer ${this.sfToken}` } }
  );

  console.log(`  Updated Salesforce: ${sfId}`);
}

/**
 * Create in Genesys (from Salesforce)
 */
```

```
async createInGenesys(sfContact) {
  const newContact = {
    firstName: sfContact.FirstName,
    lastName: sfContact.LastName,
    email: sfContact.Email,
    phoneNumbers: sfContact.Phone ?
      [{ number: sfContact.Phone, type: 'WORK' }] : [],
    externalId: sfContact.Id
  };

  const response = await axios.post(
    `https://api.mypurecloud.com/api/v2/contacts`,
    newContact,
    { headers: { 'Authorization': `Bearer ${this.gzToken}` } }
  );

  console.log(`  Created in Genesys: ${response.data.id}`);
  this.stats.synced++;
}

/**
 * Create in Salesforce (from Genesys)
 */
async createInSalesforce(gzContact) {
  const newContact = {
    FirstName: gzContact.firstName,
    LastName: gzContact.lastName,
    Email: gzContact.email,
    Phone: gzContact.phoneNumbers?.[0]?.number,
    Last_Sync__c: new Date().toISOString()
  };

  const response = await axios.post(
    `${process.env.SALESFORCE_INSTANCE}/services/data/v57.0/sobjects/Contact`,
    newContact,
    { headers: { 'Authorization': `Bearer ${this.sfToken}` } }
  );

  // Update Genesys with SF ID (externalId)
  await this.updateGenesys(gzContact.id, { ...gzContact, Id: response.data.id });
```

```
    console.log(`  Created in Salesforce: ${response.data.id}`);
    this.stats.synced++;
  }

  /**
   * Print results
   */
  printResults() {
    console.log('\n=== SYNC RESULTS ===');
    console.log(`✓ Synced: ${this.stats.synced}`);
    console.log(`⚠ Conflicts: ${this.stats.conflicts}`);
    console.log(`✗ Errors: ${this.stats.errors}`);

    if (this.conflicts.length > 0) {
      console.log('\nConflicts (for review):');
      this.conflicts.forEach(c => {
        console.log(`  ${c.email}: ${c.winner} won`);
      });
    }
  }
}

// Usage
const sync = new BidirectionalSync(config);
await sync.runSync();
```

Deduplication Across Systems

By External ID (Best)

```
// Each contact has ID from the other system
Salesforce Contact:
  Id: 003xx000003SG
  External_Genesys_ID__c: "contact-123"
```

```
Genesys Contact:
  Id: contact-123
  externalId: "003xx000003SG"

// Match on these, never duplicate
```

By Email (Good)

```
const sfContact = sfContacts.find(c => c.Email === gzContact.email);

// Assumes email is unique per contact
// Risk: Same person with multiple emails
```

By Phone (Risky)

```
const sfContact = sfContacts.find(c => c.Phone === gzContact.phoneNumbers[0]?.number);

// Risk: Multiple people sharing phone (family, shared line)
```

Monitoring & Alerts

```
class SyncMonitor {
  constructor() {
    this.history = [];
  }

  recordSync(result) {
    this.history.push({
      timestamp: new Date(),
      synced: result.synced,
      conflicts: result.conflicts,
      errors: result.errors
    });

    // Alert if many conflicts
```

```
    if (result.conflicts > 10) {
      console.warn(`⚠ HIGH CONFLICT RATE: ${result.conflicts} conflicts`);
      this.alertOps(result);
    }

    // Alert if errors
    if (result.errors > 5) {
      console.error(`  HIGH ERROR RATE: ${result.errors} errors`);
      this.alertOps(result);
    }
  }

  getConflictRate() {
    const recent = this.history.slice(-10);
    const totalConflicts = recent.reduce((sum, h) => sum + h.conflicts, 0);
    const totalSynced = recent.reduce((sum, h) => sum + h.synced, 0);
    return totalConflicts / totalSynced;
  }
}
```

Best Practices

1. **Add Last_Sync timestamp** to both systems
   - Used to detect what changed since last sync
   - Needed for conflict detection
2. **Use External IDs**
   - SF: `External_Genesys_ID__c`
   - Genesys: `externalId`
   - Primary way to match contacts
3. **Log all conflicts**
   - Build audit trail
   - Manual review capability
   - Alerts ops team
4. **Test thoroughly**
   - What if SF and GZ both update same field?
   - What if network fails mid-sync?
   - What if token expires?
5. **Start one-way, migrate to bi-directional**
   - Less risk
   - Easier to debug

- Can add complexity later

Related Topics

- Chapter 12: Contact Sync Patterns (one-way alternative)
- Chapter 12: Activity Logging (logging sync conflicts)
- Chapter 11: Error Handling & Retry Strategy

Revision #1
Created 15 March 2026 00:45:42 by Cesar Gzz
Updated 15 March 2026 00:45:51 by Cesar Gzz