Contact Sync Patterns

Overview

Contact sync keeps Genesys and your CRM (Salesforce, Dynamics, etc.) in sync. This guide covers two approaches:

1. **One-way sync** (CRM → Genesys, simpler, recommended first)
2. **Bi-directional sync** (both ways, complex, conflict resolution needed)

Strategy 1: One-Way Sync (CRM → Genesys)

Concept

Salesforce is the **master**. Genesys is a **read-only mirror**.

```
Salesforce (Master)
    ↓ (one direction)
Genesys (Mirror)

If contact changes in SF → update Genesys
If contact changes in Genesys → ignore (no write-back)
```

Pros

- ✅ Simpler (no conflict resolution)
- ✅ Faster (no locking)
- ✅ No race conditions
- ✅ Genesys data always matches SF

Cons

- □ Any changes in Genesys are lost on next sync
- □ Agents can't update CRM from Genesys

Sync Flow

```
Every 30 minutes:
  ↓
1. Query SF for changes:
   "SELECT * FROM Contact WHERE LastModifiedDate >= 30 min ago"
  ↓
2. For each contact:
   a. Check if exists in Genesys (by email, phone, external ID)
   b. If exists → PATCH (update only changed fields)
   c. If not exists → POST (create new)
  ↓
3. Log results (created, updated, skipped, errors)
  ↓
4. Alert if failures
```

Implementation

```
// sync-one-way.js

async function syncSalesforceToGenesys() {
  // 1. Fetch modified contacts from Salesforce
  const sfContacts = await querySalesforce(`
    SELECT Id, FirstName, LastName, Email, Phone
    FROM Contact
    WHERE LastModifiedDate >= LAST_N_MINUTES:30
  `);

  // 2. Get existing Genesys contacts
  const genesysContacts = await fetchGenesysContacts();
  const emailIndex = indexBy(genesysContacts, 'email');
```

```
  // 3. Process each SF contact
  let created = 0, updated = 0;

  for (const sfContact of sfContacts) {
    const genesysContact = emailIndex.get(sfContact.Email);

    if (genesysContact) {
      // Update existing
      await patchContact(genesysContact.id, {
        firstName: sfContact.FirstName,
        lastName: sfContact.LastName,
        phoneNumbers: [{ number: sfContact.Phone, type: 'WORK' }]
      });
      updated++;
    } else {
      // Create new
      await postContact({
        firstName: sfContact.FirstName,
        lastName: sfContact.LastName,
        email: sfContact.Email,
        phoneNumbers: [{ number: sfContact.Phone, type: 'WORK' }],
        externalId: sfContact.Id  // Link back
      });
      created++;
    }
  }

  console.log(`Created: ${created}, Updated: ${updated}`);
}
```

When to Use

- □ Initial implementation
- □ Salesforce is single source of truth
- □ Genesys only used for lookups/popups
- □ No need to update contacts from Genesys

Strategy 2: Bi-Directional Sync (Both Ways)

Concept

Both Salesforce AND Genesys can be updated. Sync runs both directions:

```
Salesforce ←→ Genesys


If SF updated → update Genesys
If Genesys updated → update Salesforce
If both updated at same time → conflict!
```

Pros

- ✅ Can update from both sides
- ✅ True two-way synchronization

Cons

- ❌ Much more complex
- ❌ Need conflict resolution
- ❌ Race conditions possible
- ❌ Slower (locking needed)
- ❌ Harder to debug

Conflict Resolution Strategies

Strategy A: Last-Write-Wins

Simple but risky:

```
Contact updated in both systems at same time:
  SF:     phone changed to 555-1111
  Genesys: phone changed to 555-2222
```

```
Winner: Whoever was modified LAST
Loser: Other change is overwritten
```

Problem: Unpredictable. User's change might be lost.

Strategy B: Field-Level Priority

Different fields have different sources:

```
contact_name:   SF always wins (SF is master for names)
phone_number:   Last-write-wins (allow edits in both)
tags:           Genesys always wins (agent tags)
```

Strategy C: Timestamp-Based with Locks

More robust:

```
1. Read contact with timestamp from both systems
2. Check: Who modified last?
   SF time > Genesys time → SF wins
   Genesys time > SF time → Genesys wins
3. Write winner's data to loser's system
4. Log conflict for human review
```

Implementation

```
// sync-bidirectional.js

async function syncBidirectional() {
  // 1. Fetch from both systems
  const sfContacts = await querySalesforce(`
    SELECT Id, FirstName, LastName, Email, LastModifiedDate
    FROM Contact
  `);

  const genesysContacts = await fetchGenesysContacts();

  // 2. Index for matching
```

```
const emailIndex = new Map();
for (const contact of [...sfContacts, ...genesysContacts]) {
  if (!emailIndex.has(contact.email)) {
    emailIndex.set(contact.email, []);
  }
  emailIndex.get(contact.email).push(contact);
}

// 3. Detect conflicts and sync
let conflicts = [];

for (const [email, records] of emailIndex) {
  if (records.length === 1) {
    // Only in one system, create in other
    await syncOneWay(records[0]);
  } else if (records.length === 2) {
    // In both systems, detect conflict
    const sf = records.find(r => r.system === 'salesforce');
    const gz = records.find(r => r.system === 'genesys');

    const winner = resolveConflict(sf, gz);

    if (winner === sf) {
      // SF wins, update Genesys
      await updateGenesys(gz.id, sf.data);
    } else {
      // Genesys wins, update SF
      await updateSalesforce(sf.id, gz.data);
    }

    // Log for audit
    if (sf.lastModifiedDate !== gz.dateModified) {
      conflicts.push({ email, sf, gz, winner });
    }
  }
}

// 4. Alert on conflicts
if (conflicts.length > 0) {
```

```
    await alertConflicts(conflicts);
  }
}

function resolveConflict(sfContact, gzContact) {
  // Last-write-wins
  const sfTime = new Date(sfContact.LastModifiedDate);
  const gzTime = new Date(gzContact.dateModified);

  if (sfTime > gzTime) {
    console.log(`Conflict: SF wins for ${sfContact.Email}`);
    return sfContact;
  } else {
    console.log(`Conflict: Genesys wins for ${sfContact.Email}`);
    return gzContact;
  }
}
```

Conflict Resolution Workflow

```
Conflict Detected
    ↓
Log both versions (SF and Genesys)
    ↓
Apply resolution strategy
    ├─ Last-write-wins?
    ├─ Field-level priority?
    └─ Manual approval?
    ↓
Update loser system
    ↓
Send alert to admin
    ↓
Update audit log
```

Data Deduplication

Problem: Same person, multiple records

```
Salesforce:
  Record 1: john.doe@example.com
  Record 2: JDoe@example.com (same person)

Genesys would create 2 records!
```

Solution: Match on Multiple Fields

```
function findMatchingContact(sfContact, genesysContacts) {
  // Try email (best match)
  let match = genesysContacts.find(
    gc => gc.email?.toLowerCase() === sfContact.Email.toLowerCase()
  );
  if (match) return match;

  // Try phone (second best)
  match = genesysContacts.find(
    gc => gc.phoneNumbers?.[0]?.number === sfContact.Phone
  );
  if (match) return match;

  // Try first + last name (risky, could be duplicate)
  match = genesysContacts.find(
    gc => gc.firstName === sfContact.FirstName &&
          gc.lastName === sfContact.LastName &&
          gc.externalOrganization?.id === sfContact.AccountId
  );
  if (match) return match;

  // Not found
  return null;
}
```

Best Practice: External IDs

Link records across systems:

```
Salesforce Contact:
  ID: 003xx000003SG
  Email: john@example.com

Genesys Contact:
  ID: contact-123
  externalId: "003xx000003SG"  ← Link back
  email: john@example.com
```

On next sync, use `externalId` to find exact match.

Field Mapping Reference

Salesforce	Genesys	Sync Direction	Conflicts
`Contact.Id`	`externalId`	SF → GZ	SF (master)
`Contact.FirstName`	`firstName`	Both	SF (master)
`Contact.LastName`	`lastName`	Both	SF (master)
`Contact.Email`	`email`	Both	Last-write
`Contact.Phone`	`phoneNumbers[0]`	Both	Last-write
`Contact.AccountId`	`org.externalId`	SF → GZ	SF
`Contact.LeadScore`	N/A	SF → GZ	N/A
`Contact.Tags` (custom)	`attributes.tags`	Both	GZ wins

Sync Frequency Trade-offs

Frequency	Pros	Cons	Use Case
Every 5 min	Real-time	High API load, cost	Critical data
Every 15 min	Near real-time	Medium load	Normal ops
Every 30 min	Balanced	Slight delay	Standard
Hourly	Low cost	1hr delay	Low priority

Frequency	Pros	Cons	Use Case
Daily	Very low cost	24hr delay	Batch jobs

Recommendation: Start with 30 minutes, adjust based on needs.

Monitoring Sync Health

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

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

    // Alert if too many errors
    if (result.errors.length > 10) {
      this.alertHighErrorRate(result);
    }

    // Alert if sync slow
    if (result.duration > 5 * 60 * 1000) { // 5 minutes
      this.alertSlowSync(result);
    }
  }

  getHealthScore() {
    const recent = this.history.slice(-10); // Last 10 syncs
    const avgErrors = recent.reduce((sum, h) => sum + h.errors.length, 0) / recent.length;
    const avgDuration = recent.reduce((sum, h) => sum + h.duration, 0) / recent.length;
```

```
    const score = 100 - (avgErrors * 5) - (avgDuration / 1000); // Deduct for errors and slow
    return Math.max(0, score);
  }
}
```

When to Use Which Strategy

One-Way Sync (Recommended First)

Use if:

- ✅ CRM is single source of truth
- ✅ Only need lookups in Genesys
- ✅ No updates from agent desktop
- ✅ Simple, maintainable

Bi-Directional Sync

Use if:

- ✅ Agents need to update contacts in Genesys
- ✅ Those updates must sync back to CRM
- ✅ Can handle complexity
- ✅ Have conflict resolution strategy

Migration Path

Phase 1: One-Way

- Sync SF → Genesys (read-only)
- Test with small set (100 contacts)
- Verify data accuracy

Phase 2: Expand

- Sync all contacts
- Run for 2 weeks, monitor
- Ensure no data loss

Phase 3: Bi-Directional

- Add reverse sync (Genesys → SF)
- Implement conflict resolution
- Test extensively
- Monitor for conflicts

Phase 4: Optimize

- Tune frequency
- Add caching
- Optimize performance
- Monitor cost

Related Topics

- Chapter 11: Real-World Contact Sync Example
- Chapter 11: API Endpoints Reference (Contacts API)
- Chapter 12: Activity Logging (logging interactions back)

Revision #1
Created 15 March 2026 00:45:02 by Cesar Gzz
Updated 15 March 2026 00:45:13 by Cesar Gzz