GDPR & Data Governance in CRM Integration

Overview

When syncing contact data between Genesys and Salesforce, you handle personal data (PII - Personally Identifiable Information). GDPR, CCPA, and similar regulations require proper handling, deletion, and consent management.

Key Regulations:

- **GDPR (EU):** Right to be forgotten, consent required
- **CCPA (California):** Right to deletion, right to know
- **PIPEDA (Canada):** Similar to GDPR
- **Your Local Laws:** May have additional requirements

PII Data Types

What's PII?

Information that can identify a person:

```
✓ PII - Don't store longer than needed:
  - Full name
  - Email address
  - Phone number
  - Address
  - Social Security Number
  - Payment card info
  - Biometric data
  - IP address + timestamp
```

```
✗ NOT PII - Can store longer:
  - Company name
  - Job title
  - Aggregated analytics (no individuals)
  - Anonymized data (truly de-identified)
```

GDPR Right to Deletion ("Right to be Forgotten")

Requirement

Customer asks: "Delete all my data"

You MUST:

1. Delete from Salesforce
2. Delete from Genesys
3. Delete from call recordings
4. Delete from transcripts
5. Delete from logs/backups (after retention period)

Implementation

```
/**
 * GDPR: Delete all data for a contact
 */
async function deleteContactCompletelyGDPR(email) {
  console.log(`□□ GDPR Deletion: ${email}`);

  try {
    // 1. Find contact in both systems
    const sfContact = await findSalesforceContactByEmail(email);
    const gzContact = await findGenesysContactByEmail(email);
```

```
if (!sfContact && !gzContact) {
  console.log(`  No contact found for ${email}`);
  return { status: 'not_found' };
}

// 2. Delete from Salesforce
if (sfContact) {
  console.log(`  Deleting Salesforce contact: ${sfContact.Id}`);
  await axios.delete(
    `${process.env.SALESFORCE_INSTANCE}/services/data/v57.0/sobjects/Contact/${sfContact.Id}`,
    { headers: { 'Authorization': `Bearer ${this.sfToken}` } }
  );

  // Also delete associated records
  await deleteAssociatedSalesforceRecords(sfContact.Id);
}

// 3. Delete from Genesys
if (gzContact) {
  console.log(`  Deleting Genesys contact: ${gzContact.id}`);
  await axios.delete(
    `https://api.mypurecloud.com/api/v2/contacts/${gzContact.id}`,
    { headers: { 'Authorization': `Bearer ${this.gzToken}` } }
  );
}

// 4. Find and delete recordings (if applicable)
const recordings = await findRecordingsByPhone(sfContact?.Phone);
for (const recording of recordings) {
  console.log(`  Deleting recording: ${recording.id}`);
  await deleteRecording(recording.id);
}

// 5. Find and delete call logs
const callLogs = await findCallLogsByEmail(email);
for (const log of callLogs) {
  console.log(`  Deleting call log: ${log.id}`);
  await deleteCallLog(log.id);
}
```

```
    // 6. Log the deletion (for compliance audit)
    await logGDPRDeletion({
      email,
      deletedAt: new Date(),
      deletedFrom: ['salesforce', 'genesys', 'recordings', 'call_logs'],
      reason: 'GDPR Right to Deletion'
    });

    console.log(`✅ Completely deleted: ${email}`);
    return { status: 'deleted', deletedFrom: ['salesforce', 'genesys'] };

  } catch (error) {
    console.error(`❌ Deletion failed: ${error.message}`);
    throw new Error(`Failed to delete ${email}: ${error.message}`);
  }
}

/**
 * Delete associated Salesforce records (tasks, events, etc.)
 */
async function deleteAssociatedSalesforceRecords(contactId) {
  // Delete tasks
  const tasks = await querySalesforce(`
    SELECT Id FROM Task
    WHERE WhoId = '${contactId}'
  `);

  for (const task of tasks.records) {
    await axios.delete(
      `${process.env.SALESFORCE_INSTANCE}/services/data/v57.0/sobjects/Task/${task.Id}`,
      { headers: { 'Authorization': `Bearer ${this.sfToken}` } }
    );
  }

  // Delete events
  const events = await querySalesforce(`
    SELECT Id FROM Event
    WHERE WhoId = '${contactId}'
  `);
```

```
  for (const event of events.records) {
    await axios.delete(
      `${process.env.SALESFORCE_INSTANCE}/services/data/v57.0/sobjects/Event/${event.Id}`,
      { headers: { 'Authorization': `Bearer ${this.sfToken}` } }
    );
  }

  console.log(`  Deleted ${tasks.records.length} tasks and ${events.records.length} events`);
}
```

Consent Management

Consent Requirements

Before storing/processing PII, you need:

- ✓ Explicit consent (not implied)
- ✓ Clear description of what data you collect
- ✓ Clear description of how you use it
- ✓ Easy way to withdraw consent
- ✓ Record of when consent was given

Implementation

```
/**
 * Create contact with consent tracking
 */
async function createContactWithConsent(contactData, consentInfo) {
  // 1. Verify consent was given
  if (!consentInfo.consentGiven) {
    throw new Error('Cannot create contact without explicit consent');
  }

  // 2. Create contact
  const contact = await axios.post(
```

```
    `https://api.mypurecloud.com/api/v2/contacts`,
    {
      firstName: contactData.firstName,
      lastName: contactData.lastName,
      email: contactData.email,
      phoneNumbers: contactData.phoneNumbers
    },
    { headers: { 'Authorization': `Bearer ${this.gzToken}` } }
  );

  // 3. Store consent record in Salesforce
  const consentRecord = {
    Contact_Id__c: contact.id,
    Email: contactData.email,
    Consent_Given__c: true,
    Consent_Date__c: new Date().toISOString(),
    Consent_Type__c: consentInfo.type, // 'marketing', 'service', etc.
    Consent_Channel__c: consentInfo.channel, // 'email', 'phone', 'web'
    Consent_Version__c: consentInfo.version
  };

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

  // 4. Log for audit
  console.log(`✓ Contact created with consent: ${contact.id}`);

  return contact;
}

/**
 * Withdraw consent
 */
async function withdrawConsent(email) {
  console.log(`Withdrawing consent for: ${email}`);

  // Find consent records
```

```
  const consents = await querySalesforce(`
    SELECT Id FROM Contact_Consent__c
    WHERE Email = '${email}'
  `);

  // Update all to withdrawn
  for (const consent of consents.records) {
    await axios.patch(
      `${process.env.SALESFORCE_INSTANCE}/services/data/v57.0/sobjects/Contact_Consent__c/${consent.Id}`,
      {
        Consent_Withdrawn__c: true,
        Consent_Withdrawn_Date__c: new Date().toISOString()
      },
      { headers: { 'Authorization': `Bearer ${this.sfToken}` } }
    );
  }

  console.log(`✓ Consent withdrawn for: ${email}`);
}
```

Call Recording Retention

Legal Retention Requirements

```
Recording Retention Rules:
  ├─ Default: 7-10 years (varies by jurisdiction)
  ├─ Legal hold: Keep indefinitely if in dispute
  ├─ Customer deleted: Delete after 30 days if requested
  └─ End of retention: Delete automatically
```

Implementation

```
/**
 * Check if recording should be kept or deleted
 */
```

```
async function evaluateRecordingRetention(recording) {
  const createdDate = new Date(recording.dateCreated);
  const now = new Date();
  const ageInYears = (now - createdDate) / (1000 * 60 * 60 * 24 * 365);

  // 1. Is this contact deleted per GDPR?
  const isDeletionRequested = await checkGDPRDeletionRequest(recording.contact);
  if (isDeletionRequested) {
    console.log(`Deleting recording (GDPR): ${recording.id}`);
    await deleteRecording(recording.id);
    return 'deleted_gdpr';
  }

  // 2. Is this on legal hold?
  const isOnLegalHold = await checkLegalHold(recording.conversation);
  if (isOnLegalHold) {
    console.log(`Keeping recording (legal hold): ${recording.id}`);
    return 'kept_legal_hold';
  }

  // 3. Has retention period expired?
  const retentionYears = 7; // Your policy
  if (ageInYears > retentionYears) {
    console.log(`Deleting recording (retention expired): ${recording.id}`);
    await deleteRecording(recording.id);
    return 'deleted_retention';
  }

  // 4. Keep recording
  console.log(`Keeping recording: ${recording.id}`);
  return 'kept';
}

/**
 * Automated daily retention check
 */
async function runDailyRetentionCheck() {
  console.log('□□ Daily Retention Check');

  const recordings = await fetchAllRecordings();
```

```
  let deleted = 0;

  for (const recording of recordings) {
    const result = await evaluateRecordingRetention(recording);
    if (result.startsWith('deleted')) {
      deleted++;
    }
  }

  console.log(`  Deleted: ${deleted} recordings`);

  // Log for compliance
  await logRetentionCheck({
    timestamp: new Date(),
    recordingsChecked: recordings.length,
    recordingsDeleted: deleted
  });
}
```

Data Masking & Anonymization

When to Mask

Mask data for:

- ✓ Sharing with third parties
- ✓ Analytics / reporting
- ✓ Development / testing environments
- ✓ Long-term archival

Never mask (keep original):

- ✓ Active customer contact
- ✓ Legal/compliance records
- ✓ Current support tickets

Implementation

```
/**
 * Mask PII for analytics
 */
function maskPIIForAnalytics(contact) {
  return {
    ...contact,
    // Keep: non-PII fields
    id: contact.id,
    createdDate: contact.createdDate,
    tier: contact.tier,
    contactCount: contact.contactCount,

    // Mask: PII fields
    firstName: 'MASKED',
    lastName: 'MASKED',
    email: 'MASKED@masked.com',
    phoneNumbers: contact.phoneNumbers?.map(p => ({
      ...p,
      number: 'XXX-XXX-' + p.number.slice(-4)  // Show last 4 digits only
    }))
  };
}

/**
 * Hash email for matching without storing
 */
function hashEmail(email) {
  const crypto = require('crypto');
  return crypto
    .createHash('sha256')
    .update(email.toLowerCase())
    .digest('hex');
}

// Usage: Match on hash, not original email
const emailHash = hashEmail(contact.email);
const matchingContact = contacts.find(c => hashEmail(c.email) === emailHash);
```

Data Breaches & Incident Response

If Data is Compromised

```
/**
 * Breach notification workflow
 */
async function handleDataBreach(affectedContacts, breachDetails) {
  console.log(`□□ DATA BREACH DETECTED`);

  try {
    // 1. Immediate actions (within 72 hours)
    console.log('1. Immediate Response:');

    // Stop the breach
    await shutdownCompromisedSystem(breachDetails.affectedSystem);

    // Assess scope
    const scope = await assessBreachScope(affectedContacts);
    console.log(`   Affected contacts: ${scope.count}`);
    console.log(`   Data exposed: ${scope.dataTypes.join(', ')}`);

    // Notify leadership
    await notifyLeadership(breachDetails);

    // 2. Regulatory notification (within required timeframe)
    console.log('2. Regulatory Notification:');

    // GDPR: Notify supervisory authority within 72 hours
    if (scope.locations.includes('EU')) {
      await notifyGDPRAuthority({
        date: new Date(),
        description: breachDetails.description,
        affectedIndividuals: scope.count,
        dataCategories: scope.dataTypes
```

```
    });
  }

  // CCPA: Notify state attorney general
  if (scope.locations.includes('California')) {
    await notifyACCPA({
      date: new Date(),
      description: breachDetails.description,
      affectedIndividuals: scope.count
    });
  }

  // 3. Notify affected individuals
  console.log('3. Individual Notification:');

  for (const contact of affectedContacts) {
    await sendBreachNotification(contact, {
      what: scope.dataTypes,
      when: breachDetails.discoveredDate,
      actions: 'Monitor credit for 1 year',
      contact: 'privacy@company.com'
    });
  }

  // 4. Document everything
  console.log('4. Documentation:');

  await createBreachReport({
    date: new Date(),
    description: breachDetails.description,
    rootCause: breachDetails.rootCause,
    affectedData: scope.dataTypes,
    affectedIndividuals: scope.count,
    remediation: breachDetails.remediation,
    preventionMeasures: breachDetails.preventionMeasures
  });

  console.log('✅ Breach handled per compliance requirements');

} catch (error) {
```

```
    console.error('□  Breach response failed:', error);
    // ESCALATE - notify compliance officer immediately
    await escalateToCompliance(error);
  }
}
```

Data Processing Agreement (DPA)

Required Documentation

If using third-party processors (Salesforce, Genesys), you need:

```
✓ Data Processing Agreement (DPA)
  - What data is processed
  - Where data is stored
  - Who has access
  - How long it's retained
  - Sub-processors used

✓ Standard Contractual Clauses (SCCs)
  - For international data transfers (EU → US)
  - Required for GDPR compliance

✓ Privacy Impact Assessment (PIA)
  - Document risks
  - Mitigation measures
  - Legal basis for processing
```

Compliance Checklist

- ☐ Privacy Policy updated (what data you collect, why, how long)
- ☐ Consent collected before storing PII
- ☐ Data Processing Agreement signed (with Salesforce, Genesys)
- ☐ Standard Contractual Clauses for international transfers

- ☐ Retention policy documented (how long to keep data)
- ☐ Deletion procedure documented (how to remove on request)
- ☐ Encryption in transit (HTTPS, OAuth tokens)
- ☐ Encryption at rest (Salesforce, Genesys encryption)
- ☐ Access controls (who can view data)
- ☐ Audit logging (who accessed what, when)
- ☐ Breach response plan (what to do if compromised)
- ☐ Staff training (privacy, data security)
- ☐ Regular audits (third-party or internal)

Best Practices

1. **Minimize data collection**
   - Only collect what you need
   - Delete when no longer needed
2. **Encrypt everything**
   - Data in transit (HTTPS)
   - Data at rest (DB encryption)
   - Backups (encrypted backup systems)
3. **Access controls**
   - Principle of least privilege
   - Only those who need access get it
   - Remove access when no longer needed
4. **Audit logging**
   - Log all access to PII
   - Log all modifications
   - Log all deletions
   - Keep audit logs for compliance period
5. **Regular reviews**
   - Review who has access
   - Review what data you're storing
   - Review retention policies
   - Review third-party access

Related Topics

- Chapter 12: Contact Sync Patterns

- Chapter 12: Activity Logging & Webhooks
- Chapter 11: API Endpoints Reference (data APIs)

Revision #1
Created 15 March 2026 00:46:01 by Cesar Gzz
Updated 15 March 2026 00:46:09 by Cesar Gzz