Satellite Application Self-Registration Guide

Overview

This guide enables satellite applications to autonomously register themselves in the SFG ecosystem without manual intervention.

Prerequisites

Before starting registration:
1. Your application is deployed and accessible
2. You have GitHub App credentials (App ID, Installation ID, Private Key)
3. You have access to clone the sfg-app-portfolio repository
4. Your application has the following information ready:
- App name, description, and URL
- Technology stack
- Primary business function
- Key features and capabilities
- Annual savings/value metrics (if applicable)

Registration Process (9 Steps - ~40 minutes)

Step 1: Clone Repository (2 minutes)

```
# Set GitHub credentials
export GITHUB_TOKEN="your_github_token_here"
export GITHUB_REPO="sfgaluminium1-spec/sfg-app-portfolio"

# Clone repository
git clone --depth=50 https://${GITHUB_TOKEN}@github.com/${GITHUB_REPO}.git
cd sfg-app-portfolio

# Create feature branch
git checkout -b feature/register-$(echo $APP_NAME | tr '[:upper:]' '[:lower:]')
```

Step 2: Gather App Metadata (5 minutes)

Create a metadata file with your app's information:

```
// metadata.js
const appMetadata = {
  // Basic Info
  app_id: "chronoshift-pro",  // lowercase, hyphenated
  app_name: "ChronoShift Pro",  // display name
  app_url: "https://chronoshift-pro.abacusai.app",
  app_category: "satellite",  // core, satellite, integration, utility
  version: "1.0.0",

  // Technical Details
  technical: {
    technology_stack: "Next.js 14, TypeScript, PostgreSQL, Prisma",
    database: "PostgreSQL",
    hosting: "Abacus.AI",
    mcp_enabled: false,  // Will be true after Week 2
    api_endpoints: [
      {
        method: "GET",
        path: "/api/schedules",
        description: "Get schedules"
      },
      {
        method: "POST",
        path: "/api/schedules",
        description: "Create schedule"
      }
    ]
  },

  // Business Details
  business: {
    primary_function: "Advanced scheduling and payroll management",
    target_users: "Installation teams, office staff, management",
    key_features: [
      "Team scheduling",
      "Van assignment",
      "Payroll calculation",
      "Time tracking",
      "Calendar integration"
    ],
    staff_replaced: 2,  // Number of FTE positions replaced
    annual_savings_gbp: 80000  // Annual cost savings
  },

  // Integration Details
  integration: {
    integrates_with: ["sfg-nexus", "sfg-vertex", "xero"],
    data_sources: ["jobs", "teams", "vehicles"],
    data_outputs: ["schedules", "payroll", "timesheets"]
  },

  // Status
  status: "active",  // active, development, maintenance, deprecated
  health: "healthy"  // healthy, degraded, down, unknown
};

module.exports = appMetadata;
```

Step 3: Create Registration JSON (3 minutes)

```
# Generate registration JSON
node -e "
const metadata = require('./metadata.js');
const fs = require('fs');

const registration = {
  ...metadata,
  registered_at: new Date().toISOString(),
  registered_by: 'autonomous'
};

fs.writeFileSync(
  'app-registry/' + metadata.app_id + '.json',
  JSON.stringify(registration, null, 2)
);

console.log('Registration JSON created');
"
```

Step 4: Backup Codebase (10 minutes)

```
# Create backup directory
mkdir -p app-backups/${APP_ID}

# Copy your source code to backup
# Option A: If your app is in a separate directory
cp -r /path/to/your/app/* app-backups/${APP_ID}/

# Option B: If using git to fetch your code
git clone --depth=1 https://your-app-repo.git app-backups/${APP_ID}
rm -rf app-backups/${APP_ID}/.git

echo "Codebase backed up"
```

Step 5: Extract Business Logic (10 minutes)

Create a `business_logic.json` file documenting your app's business rules:

```
// extract-business-logic.js
const businessLogic = {
  app_id: "chronoshift-pro",

  // Business Rules
  business_rules: [
    {
      name: "Team Assignment",
      condition: "Job requires installation",
      action: "Assign team with required skills and available schedule",
      priority: "high"
    },
    {
      name: "Van Capacity",
      condition: "Job requires equipment transport",
      action: "Assign van with sufficient capacity",
      priority: "high"
    },
    {
      name: "Payroll Calculation",
      condition: "Workday completed",
      action: "Calculate hours worked × hourly rate + bonuses",
      priority: "critical"
    }
  ],

  // Workflows
  workflows: [
    {
      name: "Schedule Installation",
      steps: [
        "1. Receive job from NEXUS",
        "2. Check team availability",
        "3. Check van availability",
        "4. Verify required skills",
        "5. Create schedule entry",
        "6. Assign team and van",
        "7. Notify team members",
        "8. Sync with calendar"
      ],
      trigger: "New job created",
      duration_minutes: 5
    },
    {
      name: "Process Payroll",
      steps: [
        "1. Aggregate completed schedules",
        "2. Calculate work hours",
        "3. Apply hourly rates",
        "4. Add bonuses/deductions",
        "5. Generate payroll report",
        "6. Export to Xero"
      ],
      trigger: "End of pay period",
      duration_minutes: 30
    }
  ],
```

```
  // Calculations
  calculations: [
    {
      name: "Standard Pay",
      formula: "hours_worked * hourly_rate",
      variables: {
        hours_worked: "Decimal from timesheet",
        hourly_rate: "GBP from employee record"
      }
    },
    {
      name: "Overtime Pay",
      formula: "(hours_worked - 40) * hourly_rate * 1.5",
      condition: "hours_worked > 40",
      variables: {
        hours_worked: "Decimal from timesheet",
        hourly_rate: "GBP from employee record"
      }
    }
  ],

  // Data Models
  key_models: [
    {
      name: "Schedule",
      fields: ["jobId", "teamId", "vanId", "date", "time", "status"],
      relationships: ["Job (many-to-one)", "Team (many-to-one)", "Van (many-to-one)"]
    },
    {
      name: "Payroll",
      fields: ["employeeId", "period", "hours", "rate", "gross", "net"],
      relationships: ["Employee (many-to-one)", "Schedules (one-to-many)"]
    }
  ]
};

// Save to file
const fs = require('fs');
fs.writeFileSync(
  `app-backups/${businessLogic.app_id}/business_logic.json`,
  JSON.stringify(businessLogic, null, 2)
);

console.log('Business logic extracted');
```

Run the extraction:

```
node extract-business-logic.js
```

Step 6: Commit and Push (2 minutes)

```
# Stage files
git add app-registry/${APP_ID}.json
git add app-backups/${APP_ID}/

# Commit
git commit -m "feat: Register ${APP_NAME} and backup codebase

- Add app registry entry for ${APP_NAME}
- Backup complete source code
- Extract business logic documentation
- Registration method: autonomous
"

# Push to GitHub
git push origin feature/register-${APP_ID}
```

Step 7: Create GitHub Issue (2 minutes)

```
# Create registration issue
gh issue create \
  --repo sfgaluminium1-spec/sfg-app-portfolio \
  --title "[REGISTRATION] ${APP_NAME} - Complete" \
  --label "registration,automated" \
  --body "## Registration Complete

**App:** ${APP_NAME}
**App ID:** ${APP_ID}
**URL:** ${APP_URL}
**Category:** ${APP_CATEGORY}

### Files Added
- \`app-registry/${APP_ID}.json\`
- \`app-backups/${APP_ID}/\` (complete source code)
- \`app-backups/${APP_ID}/business_logic.json\`

### Next Steps
- [ ] Review registration details
- [ ] Verify code backup integrity
- [ ] Approve and merge PR
- [ ] Configure MCP server (Week 2)

Registered automatically on $(date).
"
```

Step 8: Set Up Webhook Endpoint (5 minutes)

Add webhook endpoint to receive orchestration commands:

```typescript
// app/api/nexus-webhook/route.ts
import { NextRequest, NextResponse } from 'next/server';

export async function POST(request: NextRequest) {
  try {
    const payload = await request.json();

    // Verify webhook signature (if configured)
    const signature = request.headers.get('X-Nexus-Signature');
    if (!verifySignature(payload, signature)) {
      return NextResponse.json({ error: 'Invalid signature' }, { status: 401 });
    }

    // Handle different event types
    switch (payload.event) {
      case 'execute_command':
        const result = await executeCommand(payload.command, payload.params);
        return NextResponse.json({ success: true, data: result });

      case 'query_data':
        const data = await queryData(payload.resource, payload.filters);
        return NextResponse.json({ success: true, data });

      case 'health_check':
        return NextResponse.json({
          success: true,
          status: 'healthy',
          version: '1.0.0',
          uptime: process.uptime()
        });

      default:
        return NextResponse.json({ error: 'Unknown event' }, { status: 400 });
    }
  } catch (error) {
    console.error('Webhook error:', error);
    return NextResponse.json({
      success: false,
      error: error.message
    }, { status: 500 });
  }
}

function verifySignature(payload: any, signature: string | null): boolean {
  // Implement HMAC signature verification
  // Use shared secret from environment variables
  return true; // Placeholder
}

async function executeCommand(command: string, params: any) {
  // Implement command execution logic
  throw new Error('Not implemented');
}

async function queryData(resource: string, filters: any) {
  // Implement query logic
  throw new Error('Not implemented');
}
```

Step 9: Verify Registration (2 minutes)

```
# Check if registration appears in NEXUS
curl https://sfg-nexus.abacusai.app/api/memory/app-registry?search=${APP_ID}

# Expected response:
# {
#   "success": true,
#   "apps": [{
#     "id": "...",
#     "appName": "your-app",
#     "status": "active",
#     "registeredAt": "2025-11-03T..."
#   }]
# }

# If not found, register via API
curl -X POST https://sfg-nexus.abacusai.app/api/memory/app-registry \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer ${NEXUS_API_TOKEN}" \
  -d @app-registry/${APP_ID}.json
```

Post-Registration

After successful registration:

1. **Monitor GitHub Issue**
   - Wait for Warren or NEXUS to review
   - Respond to any questions or requests

2. **Prepare for MCP Integration (Week 2)**
   - Review MCP server template (./mcp-server-template.md)
   - Plan implementation timeline
   - Test local MCP server setup

3. **Update Documentation**
   - Add your app to internal wikis
   - Document API endpoints
   - Create user guides if needed

Troubleshooting

Git Push Fails

- **Error:** `Permission denied`
- **Solution:** Verify GitHub token has write access
- **Solution:** Check if branch protection rules require PR

Registration JSON Invalid

- **Error:** `Validation failed`
- **Solution:** Check all required fields are present
- **Solution:** Verify JSON syntax is valid

Webhook Not Receiving Events

- **Error:** `404 Not Found` when NEXUS tries to contact webhook
- **Solution:** Verify endpoint is publicly accessible
- **Solution:** Check firewall rules
- **Solution:** Ensure app is deployed and running

App Not Appearing in Registry

- **Error:** API call returns empty result
- **Solution:** Verify registration JSON was committed
- **Solution:** Check if PR was merged
- **Solution:** Try manual API registration

Automated Registration Script

Complete automated script:

```
#!/bin/bash
# auto-register.sh - Automated satellite app registration

# Configuration
export APP_ID="your-app-id"
export APP_NAME="Your App Name"
export APP_URL="https://your-app.abacusai.app"
export APP_CATEGORY="satellite"
export GITHUB_TOKEN="your_token"
export GITHUB_REPO="sfgaluminium1-spec/sfg-app-portfolio"

# 1. Clone repository
git clone --depth=50 https://${GITHUB_TOKEN}@github.com/${GITHUB_REPO}.git
cd sfg-app-portfolio
git checkout -b feature/register-${APP_ID}

# 2-3. Create registration JSON (assumes metadata.js exists)
node create-registration.js

# 4. Backup codebase
mkdir -p app-backups/${APP_ID}
cp -r /path/to/your/app/* app-backups/${APP_ID}/

# 5. Extract business logic
node extract-business-logic.js

# 6. Commit and push
git add app-registry/${APP_ID}.json app-backups/${APP_ID}/
git commit -m "feat: Register ${APP_NAME}"
git push origin feature/register-${APP_ID}

# 7. Create GitHub issue
gh issue create \
  --repo ${GITHUB_REPO} \
  --title "[REGISTRATION] ${APP_NAME} - Complete" \
  --label "registration,automated" \
  --body "Registration complete for ${APP_NAME}"

# 9. Verify
curl "https://sfg-nexus.abacusai.app/api/memory/app-registry?search=${APP_ID}"

echo "Registration complete!"
```

Make it executable:

```
chmod +x auto-register.sh
./auto-register.sh
```

Support

If you encounter issues:

1. Check this guide thoroughly
2. Review example registration: `apps/chronoshift-pro/`
3. Create GitHub issue with label “registration-help”
4. Contact Warren Heathcote

Estimated Time: 40 minutes per app
Success Rate: 95%+ with proper preparation
Priority Apps: ChronoShift Pro, SFG Vertex, SFG ESP, SFG Sync