Southern Haulers Data Registry Guide

Overview

The Southern Haulers Data Registry is a comprehensive, TypeScript-based content management system that serves as the **single source of truth** for all content across the website and application. This registry architecture centralizes all data—from port information to service descriptions, testimonials to FAQs—making content management, updates, and consistency effortless.

Architecture

Registry Structure

```
apps/web/src/data/
├── registry.ts          # Master registry with all exports and utilities
├── ports.ts             # Port and terminal information
├── locations.ts         # Service areas and geographic coverage
├── services.ts          # Service offerings and pricing
├── features.ts          # Company features and certifications
├── testimonials.ts      # Customer testimonials and case studies
├── stats.ts             # Company statistics and milestones
└── faqs.ts              # Frequently asked questions
```

Design Principles

1. **Single Source of Truth**: All content exists in one place—no duplication across pages
2. **Type Safety**: Full TypeScript typing ensures data consistency
3. **Reusability**: Import and use data anywhere in the application
4. **Maintainability**: Update once, reflect everywhere
5. **Scalability**: Easy to add new entries without code changes
6. **Relationships**: Built-in relationships between data (services ↔ testimonials ↔ FAQs)

Registry Files

1. `ports.ts` - Ports & Terminals

Purpose: Information about all ports and terminals served by Southern Haulers.

Key Data:
- Port details (Savannah, Charleston, Jacksonville)
- Terminal information within each port
- TEU capacity and handling volumes
- Geographic coordinates
- Operational details (hours, wait times, congestion)

Types:

```
interface Port {
  id: string;
  name: string;
  code: string;
  city: string;
  state: string;
  annualTeuCapacity: number;
  terminals: Terminal[];
  // ... more fields
}
```

Example Usage:

```
import { PORTS, getPortById } from '@/data/ports';

const savannahPort = getPortById('savannah');
console.log(savannahPort.annualTeuHandling); // 5900000
```

---

2. `locations.ts` - Service Areas

Purpose: Geographic service coverage including cities, regions, and hub locations.

Key Data:
- Hub location (South Georgia)
- Service cities across GA, SC, FL
- Service regions (primary, secondary, extended)
- Distance from hub
- Major industries per location

Types:

```
interface Location {
  id: string;
  name: string;
  city: string;
  state: string;
  type: 'hub' | 'service-area' | 'city' | 'region';
  coordinates: { lat: number; lng: number };
  servicesAvailable: string[];
  // ... more fields
}
```

Example Usage:

```
import { getLocationsByState, getHubLocation } from '@/data/locations';

const georgiaLocations = getLocationsByState('GA');
const hub = getHubLocation();
```

---

3. `services.ts` - Service Offerings

Purpose: Complete catalog of all services with pricing, features, and descriptions.

Key Data:
- Container drayage services
- Agricultural hauling
- Warehousing & transloading
- Specialized services (reefer, hazmat, expedited)
- Pricing information
- Equipment types
- Certifications required

Types:

```
interface Service {
  id: string;
  name: string;
  category: 'drayage' | 'agricultural' | 'warehousing' | 'specialized';
  description: string;
  features: string[];
  pricing?: ServicePricing;
  // ... more fields
}
```

Example Usage:

```
import { SERVICES, getServicesByCategory } from '@/data/services';

const drayageServices = getServicesByCategory('drayage');
```

---

4. `features.ts` - Features & Capabilities

Purpose: Company features, capabilities, certifications, and competitive advantages.

Key Data:
- Technology features (GPS tracking, automated scheduling)
- Operational capabilities (storage, transloading)
- Compliance certifications (TWIC, FMCSA, C-TPAT)
- Company capabilities with metrics

Types:

```
interface Feature {
  id: string;
  name: string;
  category: 'technology' | 'operational' | 'compliance' | 'customer-service';
  description: string;
  benefits: string[];
  highlighted?: boolean;
}
```

Example Usage:

```
import { getHighlightedFeatures, CERTIFICATIONS } from '@/data/features';

const keyFeatures = getHighlightedFeatures();
```

---

5. `testimonials.ts` - Social Proof

Purpose: Customer testimonials, reviews, and detailed case studies.

Key Data:
- Customer testimonials with ratings
- Detailed case studies with metrics
- Success stories
- Industry-specific feedback
- Verified reviews

Types:

```
interface Testimonial {
  id: string;
  author: { name: string; title: string; company: string };
  rating: number;
  quote: string;
  services: string[];
  metrics?: Array<{ label: string; value: string }>;
  featured: boolean;
}
```

Example Usage:

```
import { getFeaturedTestimonials, getAverageRating } from '@/data/testimonials';

const featured = getFeaturedTestimonials();
const avgRating = getAverageRating(); // 4.9
```

---

6. `stats.ts` - Statistics & History

Purpose: Company statistics, milestones, achievements, and metrics.

Key Data:
- Operational statistics (container capacity, fleet size)
- Performance metrics (on-time delivery, customer satisfaction)
- Company milestones by year
- Awards and achievements
- Growth trends

Types:

```
interface Stat {
  id: string;
  label: string;
  value: string | number;
  category: 'operational' | 'customer' | 'growth' | 'infrastructure';
  trend?: { direction: 'up' | 'down' | 'stable'; value: string };
  featured?: boolean;
}
```

Example Usage:

```
import { getFeaturedStats, getCompanyAge, MILESTONES } from '@/data/stats';

const keyStats = getFeaturedStats();
const age = getCompanyAge(); // 15
```

---

7. `faqs.ts` - Frequently Asked Questions

Purpose: Comprehensive FAQ database organized by category.

Key Data:
- Questions and detailed answers
- Categories (services, pricing, operations, compliance, etc.)
- Related services and FAQs
- Featured FAQs for homepage
- Search functionality

Types:

```
interface FAQ {
  id: string;
  question: string;
  answer: string;
  category: 'services' | 'pricing' | 'operations' | 'compliance' | 'technology' | 'gen
eral';
  relatedServices?: string[];
  featured?: boolean;
}
```

Example Usage:

```
import { getFaqsByCategory, getFeaturedFaqs, searchFaqs } from '@/data/faqs';

const serviceFaqs = getFaqsByCategory('services');
const searchResults = searchFaqs('tracking');
```

---

8. `registry.ts` - Master Registry

Purpose: Central export point providing unified access to all registries.

Features:
- Single import for all data: `import Registry from '@/data/registry'`
- Type exports for all interfaces
- Registry statistics and metadata
- Utility functions for common operations
- Data validation
- Search across all content

Example Usage:

```
import Registry, { RegistryUtils } from '@/data/registry';

// Access any data
const allPorts = Registry.ports;
const allServices = Registry.services;

// Get related data
const serviceData = RegistryUtils.getServiceData('container-drayage');

// Search everything
const results = RegistryUtils.search('savannah');

// Get featured content
const featured = RegistryUtils.getFeaturedContent();
```

How to Use the Registry

In React Components

```
import { PORTS, getPortById } from '@/data/ports';
import { getFeaturedTestimonials } from '@/data/testimonials';

export function PortsSection() {
  return (
    <div>
      {PORTS.map(port => (
        <div key={port.id}>
          <h3>{port.name}</h3>
          <p>{port.description}</p>
          <span>{port.annualTeuHandling.toLocaleString()} TEUs/year</span>
        </div>
      ))}
    </div>
  );
}
```

Dynamic Routes

```
// app/services/[slug]/page.tsx
import { getServiceBySlug } from '@/data/services';
import { getFaqsByService } from '@/data/faqs';

export async function generateStaticParams() {
  return SERVICES.map(service => ({ slug: service.slug }));
}

export default function ServicePage({ params }: { params: { slug: string } }) {
  const service = getServiceBySlug(params.slug);
  const faqs = getFaqsByService(service.id);

  return (
    <div>
      <h1>{service.name}</h1>
      <p>{service.longDescription}</p>
      {/* ... */}
    </div>
  );
}
```

For SEO and Structured Data

```
import { FAQS } from '@/data/faqs';

export function FAQSchema() {
  const schema = {
    "@context": "https://schema.org",
    "@type": "FAQPage",
    "mainEntity": FAQS.map(faq => ({
      "@type": "Question",
      "name": faq.question,
      "acceptedAnswer": {
        "@type": "Answer",
        "text": faq.answer
      }
    }))
  };

  return (
    <script type="application/ld+json">
      {JSON.stringify(schema)}
    </script>
  );
}
```

Updating the Registry

Adding a New Service

1. Open `apps/web/src/data/services.ts`
2. Add new service object to the `SERVICES` array:

```
{
  id: 'new-service',
  name: 'New Service Name',
  shortName: 'Short Name',
  category: 'drayage', // or 'agricultural', 'warehousing', 'specialized'
  slug: 'new-service',
  description: 'Short description',
  longDescription: 'Detailed description',
  features: [
    'Feature 1',
    'Feature 2',
  ],
  benefits: [
    'Benefit 1',
    'Benefit 2',
  ],
  availability: 'all-locations',
  icon: 'Package',
}
```

1. Service automatically appears in:
   - Services listing pages
   - Navigation menus
   - Related sections
   - Search results

Adding a New Testimonial

1. Open `apps/web/src/data/testimonials.ts`
2. Add to the `TESTIMONIALS` array:

```
{
  id: 'testimonial-new',
  author: {
    name: 'Customer Name',
    title: 'Job Title',
    company: 'Company Name',
    industry: 'Industry Type',
    location: 'City, State',
  },
  rating: 5,
  quote: 'Short impactful quote',
  fullText: 'Complete testimonial text',
  services: ['service-id-1', 'service-id-2'], // Reference service IDs
  date: '2024-10-28',
  featured: true, // Show on homepage
  verified: true,
}
```

Adding a New FAQ

1. Open `apps/web/src/data/faqs.ts`
2. Add to the `FAQS` array:

```
{
  id: 'faq-new',
  question: 'What is your question?',
  answer: 'Detailed answer with specifics',
  category: 'services', // or 'pricing', 'operations', 'compliance', 'technology',
'general'
  relatedServices: ['service-id'], // Optional
  relatedFaqs: ['other-faq-id'], // Optional
  featured: true, // Show on homepage
}
```

Adding a New Port/Terminal

1. Open `apps/web/src/data/ports.ts`
2. Add to the `PORTS` array with all terminals:

```
{
  id: 'new-port',
  name: 'Port of New City',
  displayName: 'Port of New City, ST',
  code: 'NCT',
  city: 'New City',
  state: 'State',
  stateCode: 'ST',
  region: 'Southeast',
  coordinates: { lat: 30.0000, lng: -80.0000 },
  annualTeuCapacity: 1000000,
  annualTeuHandling: 900000,
  rank: 10,
  description: 'Port description',
  terminals: [
    {
      id: 'terminal-1',
      name: 'Main Terminal',
      code: 'MT',
      location: 'New City, ST',
      operator: 'Port Authority',
      features: ['Feature 1', 'Feature 2'],
    },
  ],
  features: ['TWIC certified', '24/7 operations'],
  website: 'https://port.com',
}
```

Adding a New Location

1. Open `apps/web/src/data/locations.ts`
2. Add to the `LOCATIONS` array:

```
{
  id: 'new-city',
  name: 'New City',
  displayName: 'New City, ST',
  city: 'New City',
  state: 'State',
  stateCode: 'ST',
  region: 'Southeast',
  coordinates: { lat: 30.0000, lng: -80.0000 },
  type: 'city', // or 'hub', 'service-area', 'region'
  population: 100000,
  description: 'City description',
  servicesAvailable: ['container-drayage', 'agricultural-hauling'],
  distanceFromHub: { miles: 100, hours: 1.5 },
  majorIndustries: ['Manufacturing', 'Distribution'],
}
```

Updating Statistics

1. Open `apps/web/src/data/stats.ts`
2. Update the relevant stat value:

```
{
  id: 'annual-moves',
  value: '18,000+', // Update this value
  trend: {
    direction: 'up',
    value: '25%', // Update growth percentage
    period: 'YoY',
  },
}
```

Best Practices

1. Consistent IDs

- Use kebab-case for all IDs: `container-drayage`, `port-savannah`
- Make IDs descriptive and unique
- Never change existing IDs (breaks references)

2. Complete Data

- Fill in all required fields
- Add optional fields when available
- Include descriptions for clarity
- Provide metrics when possible

3. Relationships

- Reference related items by ID
- Use `relatedServices`, `relatedFaqs`, etc.
- Maintain bidirectional relationships

4. Featured Content

- Mark important items as `featured: true`

- Featured items appear on homepage
- Limit featured items to most impactful

5. Keep Data Current

- Update statistics annually
- Refresh testimonials regularly
- Review FAQs quarterly
- Verify port data yearly

6. Validation

Run validation to check data integrity:

```
import { RegistryUtils } from '@/data/registry';

const validation = RegistryUtils.validateRegistry();
if (!validation.valid) {
  console.error('Registry errors:', validation.errors);
}
```

Common Patterns

Building a Service Page

```
import { getServiceBySlug } from '@/data/services';
import { getTestimonialsByService } from '@/data/testimonials';
import { getFaqsByService } from '@/data/faqs';

export default function ServicePage({ params }: { params: { slug: string } }) {
  const service = getServiceBySlug(params.slug);
  const testimonials = getTestimonialsByService(service.id);
  const faqs = getFaqsByService(service.id);

  return (
    <>
      {/* Service details */}
      <ServiceHero service={service} />
      <FeaturesSection features={service.features} />

      {/* Related testimonials */}
      <TestimonialsSection testimonials={testimonials} />

      {/* Related FAQs */}
      <FAQSection faqs={faqs} />
    </>
  );
}
```

Building a Location Page

```
import { getLocationById } from '@/data/locations';
import { getServiceById } from '@/data/services';

export default function LocationPage({ params }: { params: { id: string } }) {
  const location = getLocationById(params.id);
  const services = location.servicesAvailable.map(id => getServiceById(id));

  return (
    <>
      <LocationHero location={location} />
      <ServicesAvailable services={services} />
      <DistanceInfo distance={location.distanceFromHub} />
    </>
  );
}
```

Search Functionality

```
import { RegistryUtils } from '@/data/registry';

function SearchResults({ query }: { query: string }) {
  const results = RegistryUtils.search(query);

  return (
    <div>
      {results.services.length > 0 && (
        <ServiceResults services={results.services} />
      )}
      {results.faqs.length > 0 && (
        <FAQResults faqs={results.faqs} />
      )}
      {/* ... more result types */}
    </div>
  );
}
```

Registry Statistics

View current registry contents:

```
import { RegistryStats } from '@/data/registry';

console.log(RegistryStats);
// {
//   ports: { total: 3, terminals: 10, totalTeuCapacity: 8300000 },
//   locations: { total: 16, hubs: 1, cities: 15 },
//   services: { total: 10, byCategory: {...} },
//   testimonials: { total: 6, featured: 3, averageRating: 4.9 },
//   // ... more stats
// }
```

Troubleshooting

TypeScript Errors

Problem: Type errors when importing
Solution: Ensure you're importing from the correct path and using the right type

```
// ✅ Correct
import { Port } from '@/data/ports';
import type { Port } from '@/data/registry';

// ❌ Incorrect
import Port from '@/data/ports'; // Port is not default export
```

Missing Data

Problem: Data not appearing in components
Solution: Check that data is properly exported and imported

```
// In registry file - ensure it's in the array
export const SERVICES = [
  { id: 'my-service', /* ... */ },
];

// In component - verify import path
import { SERVICES } from '@/data/services'; // ✅
import { SERVICES } from 'data/services'; // ❌ Missing @/
```

Broken References

Problem: Related items not showing
Solution: Verify IDs match exactly (case-sensitive)

```
// Testimonial references service
services: ['container-drayage'] // Must match service ID exactly

// In services.ts
{ id: 'container-drayage', /* ... */ } // ✅ Matches
{ id: 'containerDrayage', /* ... */ } // ❌ Doesn't match
```

Future Enhancements

Potential additions to the registry:

1. **Blog/News Registry**: Articles, press releases, updates
2. **Team Registry**: Team members, roles, bios
3. **Equipment Registry**: Detailed equipment specifications
4. **Route Registry**: Specific route information and pricing
5. **Partner Registry**: Harris Brokerage and other partners
6. **Document Registry**: Downloadable forms, contracts, guides
7. **Media Registry**: Images, videos, marketing assets

Support

For questions about the data registry:
- Review this guide thoroughly
- Check TypeScript types for field requirements
- Examine existing entries as examples
- Run validation to catch errors
- Test changes in development before deploying

Last Updated: October 28, 2025
Registry Version: 1.0
Maintained By: Southern Haulers Development Team