# API Communication Patterns

## Overview

CareIQ Builder follows a specific communication pattern where all external API calls are routed through ServiceNow Scripted REST APIs. This document details the complete request/response flow, API patterns, and implementation details.

## Communication Architecture

### Three-Tier Architecture

┌─────────────────────────────────────────────────────────┐  
│ Tier 1: Client (Browser) │  
│ ┌────────────────────────────────────────────────────┐ │  
│ │ CareIQ Builder Component │ │  
│ │ - Dispatches HTTP effects │ │  
│ │ - Sends JSON payloads │ │  
│ │ - Handles responses │ │  
│ └────────────────────────────────────────────────────┘ │  
└─────────────────────────────────────────────────────────┘  
 ↓ HTTPS (JSON)  
┌─────────────────────────────────────────────────────────┐  
│ Tier 2: ServiceNow Platform │  
│ ┌────────────────────────────────────────────────────┐ │  
│ │ Scripted REST APIs │ │  
│ │ - Receive POST requests │ │  
│ │ - Extract request.body.data │ │  
│ │ - Call Script Includes │ │  
│ │ - Return JSON responses │ │  
│ └────────────────────────────────────────────────────┘ │  
│ ┌────────────────────────────────────────────────────┐ │  
│ │ CareIQ Services Script Include │ │  
│ │ - Business logic layer │ │  
│ │ - Constructs CareIQ API requests │ │  
│ │ - Handles authentication │ │  
│ │ - Transforms data │ │  
│ └────────────────────────────────────────────────────┘ │  
└─────────────────────────────────────────────────────────┘  
 ↓ HTTPS (REST)  
┌─────────────────────────────────────────────────────────┐  
│ Tier 3: CareIQ Platform (External) │  
│ - Assessment storage │  
│ - Clinical content library │  
│ - PGI management │  
│ - Returns JSON responses │  
└─────────────────────────────────────────────────────────┘

## Request Flow Pattern

### Complete Request Lifecycle

1. USER ACTION  
 User clicks "Save Question"  
 ↓  
  
2. ACTION DISPATCH  
 dispatch('SAVE\_QUESTION\_IMMEDIATELY', {questionId, changes})  
 ↓  
  
3. ACTION HANDLER  
 'SAVE\_QUESTION\_IMMEDIATELY': (coeffects) => {  
 // Validate  
 // Prepare request  
 // Dispatch effect  
 dispatch('MAKE\_UPDATE\_QUESTION\_REQUEST', {requestBody, questionId})  
 }  
 ↓  
  
4. HTTP EFFECT  
 MAKE\_UPDATE\_QUESTION\_REQUEST effect executes  
 POST to /api/x\_cadal\_careiq\_b\_0/careiq\_api/update-question  
 Body: {data: {questionId, changes}}  
 ↓  
  
5. SERVICENOW API  
 Scripted REST API receives request  
 var requestData = request.body.data;  
 Call CareIQ Services Script Include  
 ↓  
  
6. SCRIPT INCLUDE  
 builderUpdateQuestion(requestData)  
 Constructs CareIQ platform request  
 Makes HTTP call to CareIQ backend  
 ↓  
  
7. CAREIQ PLATFORM  
 Processes request  
 Updates database  
 Returns response  
 ↓  
  
8. SCRIPT INCLUDE (RESPONSE)  
 Receives CareIQ response  
 Transforms data if needed  
 Returns to ServiceNow API  
 ↓  
  
9. SERVICENOW API (RESPONSE)  
 Returns JSON response to client  
 ↓  
  
10. HTTP EFFECT (SUCCESS/ERROR)  
 Effect triggers SUCCESS or ERROR action  
 'UPDATE\_QUESTION\_SUCCESS' or 'UPDATE\_QUESTION\_ERROR'  
 ↓  
  
11. SUCCESS/ERROR HANDLER  
 Updates state  
 Shows system message  
 Reloads data if needed  
 ↓  
  
12. UI RE-RENDER  
 Component reflects updated state

## Client-Side Patterns

### 1. Effect Definition (effects.js)

**Pattern**:

export const MAKE\_UPDATE\_QUESTION\_REQUEST = createHttpEffect(  
 '/api/x\_cadal\_careiq\_b\_0/careiq\_api/update-question',  
 {  
 method: 'POST',  
 dataParam: 'requestBody',  
 headers: {'Content-Type': 'application/json'},  
 startActionType: 'UPDATE\_QUESTION\_START',  
 successActionType: 'UPDATE\_QUESTION\_SUCCESS',  
 errorActionType: 'UPDATE\_QUESTION\_ERROR'  
 }  
);

**Key Points**: - **Always POST**: Even if backend uses GET, ServiceNow endpoint uses POST - **dataParam**: ‘requestBody’ - Parameter name containing request data - **Action Types**: Start, Success, and Error actions for lifecycle

### 2. Preparing Request Data

**Pattern**: Send direct fields (no data wrapper)

'SAVE\_QUESTION\_IMMEDIATELY': (coeffects) => {  
 const {action, state, dispatch} = coeffects;  
 const {questionId} = action.payload;  
  
 // Get changes from state  
 const changes = state.questionChanges[questionId];  
  
 // Prepare request body with DIRECT FIELDS (no data wrapper)  
 const requestBody = JSON.stringify({  
 questionId: questionId, // Direct field  
 label: changes.label, // Direct field  
 type: changes.type, // Direct field  
 voice: changes.voice, // Direct field  
 gt\_id: state.currentAssessmentId // Direct field  
 });  
  
 // Dispatch effect  
 dispatch('MAKE\_UPDATE\_QUESTION\_REQUEST', {  
 requestBody: requestBody,  
 questionId: questionId // Can pass meta data  
 });  
}

**Critical**: Do NOT wrap in data: {} - ServiceNow does this automatically

### 3. Handling Success Response

'UPDATE\_QUESTION\_SUCCESS': (coeffects) => {  
 const {action, state, updateState, dispatch} = coeffects;  
  
 // Extract response data  
 const response = action.payload;  
 const questionId = response.questionId || action.meta?.questionId;  
  
 // Clear change tracking  
 const updatedQuestionChanges = {...state.questionChanges};  
 delete updatedQuestionChanges[questionId];  
  
 // Clear isUnsaved flag on question  
 const updatedQuestions = state.currentQuestions.questions.map(q =>  
 q.ids.id === questionId ? {...q, isUnsaved: false} : q  
 );  
  
 updateState({  
 questionChanges: updatedQuestionChanges,  
 currentQuestions: {  
 ...state.currentQuestions,  
 questions: updatedQuestions  
 }  
 });  
  
 // Show success message  
 dispatch('ADD\_SYSTEM\_MESSAGE', {  
 type: 'success',  
 message: response.message || 'Question updated successfully!'  
 });  
  
 // Reload assessment data (post-save reload pattern)  
 if (state.currentAssessmentId) {  
 dispatch('FETCH\_ASSESSMENT\_DETAILS', {  
 assessmentId: state.currentAssessmentId  
 });  
 }  
}

### 4. Handling Error Response

'UPDATE\_QUESTION\_ERROR': (coeffects) => {  
 const {action, state, updateState, dispatch} = coeffects;  
  
 // Extract error info  
 const error = action.payload;  
 const errorMessage = error.message || error.detail || 'Failed to update question';  
  
 // Clear loading state  
 const questionId = action.meta?.questionId;  
 if (questionId) {  
 const updatedSaving = {...state.savingQuestions};  
 delete updatedSaving[questionId];  
 updateState({savingQuestions: updatedSaving});  
 }  
  
 // Show error message  
 dispatch('ADD\_SYSTEM\_MESSAGE', {  
 type: 'error',  
 message: errorMessage  
 });  
}

## ServiceNow API Patterns

### 1. Scripted REST API Structure

**File Location**: Root directory, multiple api.js files

**Naming Convention**: /api/x\_cadal\_careiq\_b\_0/careiq\_api/[operation-name]

**Example**: /api/x\_cadal\_careiq\_b\_0/careiq\_api/update-question

### 2. Standard API Template

(function process(request, response) {  
 // CRITICAL: Stash request data immediately  
 var requestData = request.body.data;  
  
 // Validate request  
 if (!requestData) {  
 response.setStatus(400);  
 response.setBody({  
 success: false,  
 message: 'No request data provided'  
 });  
 return;  
 }  
  
 // Validate required fields  
 if (!requestData.questionId) {  
 response.setStatus(400);  
 response.setBody({  
 success: false,  
 message: 'questionId is required'  
 });  
 return;  
 }  
  
 // Call Script Include with proper instantiation  
 try {  
 var careiqServices = new x\_1628056\_careiq.CareIQServices();  
 var result = careiqServices.builderUpdateQuestion(requestData);  
  
 // Return result  
 response.setStatus(200);  
 response.setBody({  
 success: true,  
 message: result.message || 'Question updated successfully',  
 data: result.data || {}  
 });  
  
 } catch (e) {  
 // SAFE ERROR HANDLING (no e.message or e.stack access)  
 var errorMsg = 'Unexpected server error occurred';  
 try {  
 if (e && typeof e.toString === 'function') {  
 errorMsg = e.toString();  
 }  
 } catch (innerE) {  
 errorMsg = 'Server error occurred';  
 }  
  
 response.setStatus(500);  
 response.setBody({  
 success: false,  
 message: errorMsg  
 });  
 }  
})(request, response);

### 3. Key ServiceNow Patterns

#### Request Data Access

**CRITICAL**: Stash request.body.data immediately

// ✅ CORRECT  
var requestData = request.body.data;  
var questionId = requestData.questionId;  
  
// ❌ WRONG - May cause consumption issues  
var questionId = request.body.data.questionId;  
var label = request.body.data.label; // Data may be consumed

**Why**: ServiceNow’s request body can only be read once

#### Data Wrapper Pattern

**ServiceNow automatically wraps POST data in .data property**:

// Client sends:  
{  
 questionId: '123',  
 label: 'Updated question'  
}  
  
// ServiceNow receives (automatically wrapped):  
{  
 data: {  
 questionId: '123',  
 label: 'Updated question'  
 }  
}  
  
// Access with:  
var requestData = request.body.data;

#### Script Include Instantiation

**CORRECT Instantiation**:

var careiqServices = new x\_1628056\_careiq.CareIQServices();

**Scope**: All CareIQ Script Includes are in x\_1628056\_careiq scope

#### Safe Error Handling

**ServiceNow Security Restriction**: Cannot access e.message or e.stack directly

// ❌ WRONG - Security violation  
catch (e) {  
 var msg = e.message; // Not allowed  
 var stack = e.stack; // Not allowed  
}  
  
// ✅ CORRECT - Safe approach  
catch (e) {  
 var errorMsg = 'Unexpected server error occurred';  
 try {  
 if (e && typeof e.toString === 'function') {  
 errorMsg = e.toString();  
 }  
 } catch (innerE) {  
 errorMsg = 'Server error occurred';  
 }  
}

### 4. Response Patterns

**Standard Success Response**:

{  
 success: true,  
 message: 'Operation completed successfully',  
 data: {  
 questionId: 'uuid-123',  
 label: 'Updated question text',  
 // ... other data  
 }  
}

**Standard Error Response**:

{  
 success: false,  
 message: 'Error description here',  
 detail: 'Additional error details',  
 code: 'ERROR\_CODE' // Optional  
}

## CareIQ Services Script Include

### 1. Script Include Structure

**File Location**: ServiceNow Script Include in x\_1628056\_careiq scope

**Class**: CareIQServices

**Source of Truth**: Delta CareIQ Services - Consolidated.js (per CLAUDE.md)

### 2. Method Pattern

builderUpdateQuestion: function(requestData) {  
 // Extract parameters  
 var questionId = requestData.questionId;  
 var label = requestData.label;  
 var type = requestData.type;  
 var voice = requestData.voice;  
 var gt\_id = requestData.gt\_id;  
  
 // Prepare CareIQ API request  
 var careiqUrl = this.getCareIQBaseUrl() + '/builder/question/' + questionId;  
 var careiqPayload = {  
 label: label,  
 type: type,  
 voice: voice,  
 guideline\_template\_id: gt\_id  
 };  
  
 // Make request to CareIQ platform  
 var request = new sn\_ws.RESTMessageV2();  
 request.setHttpMethod('PUT');  
 request.setEndpoint(careiqUrl);  
 request.setRequestHeader('Content-Type', 'application/json');  
 request.setRequestHeader('Authorization', 'Bearer ' + this.getAccessToken());  
 request.setRequestBody(JSON.stringify(careiqPayload));  
  
 var response = request.execute();  
 var responseBody = response.getBody();  
 var httpStatus = response.getStatusCode();  
  
 // Handle response  
 if (httpStatus >= 200 && httpStatus < 300) {  
 var parsedResponse = JSON.parse(responseBody);  
 return {  
 success: true,  
 message: 'Question updated successfully',  
 data: parsedResponse  
 };  
 } else {  
 return {  
 success: false,  
 message: 'Failed to update question: ' + responseBody  
 };  
 }  
}

### 3. Common Script Include Methods

**Configuration**: - getCareIQBaseUrl() - Returns CareIQ API base URL - getAccessToken() - Returns authentication token

**Assessment Operations**: - builderGetAssessments() - Fetch all assessments - builderCreateAssessment(data) - Create new assessment - builderGetSections(assessmentId) - Get sections for assessment - builderPublishAssessment(assessmentId) - Publish assessment - builderCreateVersion(assessmentId) - Create new version

**Section Operations**: - builderAddSection(data) - Add section - builderUpdateSection(sectionId, data) - Update section - builderDeleteSection(sectionId) - Delete section

**Question Operations**: - builderAddQuestionToSection(data) - Add question to section - builderUpdateQuestion(data) - Update question - builderDeleteQuestion(questionId) - Delete question

**Answer Operations**: - builderAddAnswersToQuestion(data) - Add answers to question - builderUpdateAnswer(data) - Update answer - builderDeleteAnswer(answerId) - Delete answer

**Relationship Operations**: - builderLoadAnswerRelationships(answerId) - Load relationships - builderAddBranchQuestion(data) - Add triggered question - builderDeleteBranchQuestion(relationshipId) - Delete triggered question - builderAddGuidelineRelationship(data) - Add guideline - builderDeleteGuidelineRelationship(relationshipId) - Delete guideline

**PGI Operations**: - builderLoadProblemGoals(problemId, guidelineTemplateId) - Load goals - builderAddGoal(data) - Add goal - builderLoadGoalInterventions(goalId, guidelineTemplateId) - Load interventions - builderAddIntervention(data) - Add intervention

**Search Operations**: - builderGenericTypeahead(contentType, searchText) - Generic search

## Common API Operations

### 1. Create Assessment

**Client**:

'CREATE\_NEW\_ASSESSMENT': (coeffects) => {  
 const {action, dispatch} = coeffects;  
 const {formData} = action.payload;  
  
 const requestBody = JSON.stringify({  
 guideline\_name: formData.name,  
 use\_case\_category: formData.category,  
 type: formData.type,  
 content\_source: formData.contentSource,  
 code\_policy\_number: formData.policyNumber,  
 effective\_date: formData.effectiveDate,  
 review\_date: formData.reviewDate,  
 response\_logging: formData.responseLogging  
 });  
  
 dispatch('MAKE\_CREATE\_ASSESSMENT\_REQUEST', {requestBody});  
}

**ServiceNow API**: /api/x\_cadal\_careiq\_b\_0/careiq\_api/create-assessment

**CareIQ Backend**: POST /builder/guideline-template

### 2. Add Question to Section (2-Step Pattern)

**Step 1 - Create Question**:

'ADD\_QUESTION\_TO\_SECTION': (coeffects) => {  
 const {action, state, updateState, dispatch} = coeffects;  
 const {questionData, answers} = action.payload;  
  
 // Store answers for step 2  
 updateState({  
 pendingQuestionAnswers: answers  
 });  
  
 // Create question only (no answers)  
 const requestBody = JSON.stringify({  
 section\_id: state.selectedSection,  
 label: questionData.label,  
 type: questionData.type,  
 voice: questionData.voice,  
 gt\_id: state.currentAssessmentId  
 });  
  
 dispatch('ADD\_QUESTION\_TO\_SECTION\_API', {requestBody});  
}

**ServiceNow API**: /api/x\_cadal\_careiq\_b\_0/careiq\_api/add-question-to-section

**CareIQ Backend**: POST /builder/section/{sectionId}/questions

**Step 2 - Add Answers**:

'ADD\_QUESTION\_TO\_SECTION\_SUCCESS': (coeffects) => {  
 const {action, state, updateState, dispatch} = coeffects;  
  
 const questionId = action.payload.questionId;  
  
 // Add answers if any were pending  
 if (state.pendingQuestionAnswers && state.pendingQuestionAnswers.length > 0) {  
 const requestBody = JSON.stringify({  
 question\_id: questionId,  
 answers: state.pendingQuestionAnswers  
 });  
  
 dispatch('MAKE\_ADD\_ANSWERS\_TO\_QUESTION\_REQUEST', {requestBody});  
  
 // Clear pending answers  
 updateState({  
 pendingQuestionAnswers: null  
 });  
 }  
}

**ServiceNow API**: /api/x\_cadal\_careiq\_b\_0/careiq\_api/add-answers-to-question

**CareIQ Backend**: POST /builder/question/{questionId}/answers

### 3. Load Answer Relationships

**Client**:

'LOAD\_ANSWER\_RELATIONSHIPS': (coeffects) => {  
 const {action, updateState, dispatch} = coeffects;  
 const {answerId} = action.payload;  
  
 updateState({  
 relationshipsLoading: true  
 });  
  
 const requestBody = JSON.stringify({  
 answer\_id: answerId  
 });  
  
 dispatch('MAKE\_LOAD\_ANSWER\_RELATIONSHIPS\_REQUEST', {requestBody, answerId});  
}

**ServiceNow API**: /api/x\_cadal\_careiq\_b\_0/careiq\_api/answer-relationships

**CareIQ Backend**: GET /builder/answer/{answerId}/relationships

**Response Structure**:

{  
 "success": true,  
 "data": {  
 "guidelines": [  
 {  
 "id": "guideline-uuid-1",  
 "name": "ADA Diabetes Guidelines",  
 "source": "American Diabetes Association"  
 }  
 ],  
 "triggered\_questions": [  
 {  
 "id": "question-uuid-5",  
 "label": "When were you diagnosed?",  
 "section\_id": "section-uuid-2"  
 }  
 ],  
 "problems": [  
 {  
 "id": "problem-uuid-1",  
 "name": "Uncontrolled Diabetes"  
 }  
 ],  
 "barriers": [  
 {  
 "id": "barrier-uuid-1",  
 "name": "Transportation"  
 }  
 ]  
 }  
}

### 4. Generic Typeahead Search

**Client**:

'SEARCH\_QUESTIONS': (coeffects) => {  
 const {action, state, updateState, dispatch} = coeffects;  
 const {searchText} = action.payload;  
  
 // Store context in state (reliable pattern)  
 const questionSearchContext = {  
 contentType: 'question',  
 sectionId: state.selectedSection,  
 searchText: searchText  
 };  
  
 updateState({  
 questionTypeaheadLoading: true,  
 currentQuestionSearchContext: questionSearchContext  
 });  
  
 const requestBody = JSON.stringify({  
 contentType: 'question',  
 searchText: searchText,  
 assessmentId: state.currentAssessmentId  
 });  
  
 dispatch('MAKE\_GENERIC\_TYPEAHEAD\_REQUEST', {  
 requestBody: requestBody,  
 meta: {contentType: 'question'}  
 });  
}

**ServiceNow API**: /api/x\_cadal\_careiq\_b\_0/careiq\_api/generic-typeahead

**CareIQ Backend**: GET /builder/search?type={contentType}&q={searchText}

**Success Handler (Uses Stored Context)**:

'GENERIC\_TYPEAHEAD\_SUCCESS': (coeffects) => {  
 const {action, state, updateState} = coeffects;  
 const results = action.payload.results || [];  
  
 // Use stored context, NOT meta params (meta can be undefined)  
 const questionContext = state.currentQuestionSearchContext;  
 const answerContext = state.currentAnswerSearchContext;  
 const sectionContext = state.currentSectionSearchContext;  
  
 if (questionContext && questionContext.contentType === 'question') {  
 updateState({  
 questionTypeaheadResults: results,  
 questionTypeaheadLoading: false,  
 questionTypeaheadVisible: true  
 });  
 } else if (answerContext && answerContext.contentType === 'answer') {  
 updateState({  
 answerTypeaheadResults: results,  
 answerTypeaheadLoading: false,  
 answerTypeaheadVisible: true  
 });  
 } else if (sectionContext && sectionContext.contentType === 'section') {  
 updateState({  
 sectionTypeaheadResults: results,  
 sectionTypeaheadLoading: false,  
 sectionTypeaheadVisible: true  
 });  
 }  
}

### 5. PGI Operations (State-Based Refresh)

**Load Problem Goals**:

'LOAD\_PROBLEM\_GOALS': (coeffects) => {  
 const {action, updateState, dispatch} = coeffects;  
 const {problemId, guidelineTemplateId} = action.payload;  
  
 updateState({  
 loadingProblemGoals: {  
 ...state.loadingProblemGoals,  
 [problemId]: true  
 }  
 });  
  
 const requestBody = JSON.stringify({  
 problem\_id: problemId,  
 guideline\_template\_id: guidelineTemplateId  
 });  
  
 dispatch('MAKE\_LOAD\_PROBLEM\_GOALS\_REQUEST', {  
 requestBody: requestBody,  
 problemId: problemId // Pass in meta  
 });  
}

**Add Goal to Problem (Store Context for Refresh)**:

'SAVE\_GOAL\_TO\_PROBLEM': (coeffects) => {  
 const {action, state, updateState, dispatch} = coeffects;  
 const {problemId, goalData} = action.payload;  
  
 // Store problem ID for refresh on success  
 updateState({  
 lastAddedGoalProblemId: problemId,  
 savingGoals: {  
 ...state.savingGoals,  
 [problemId]: true  
 }  
 });  
  
 const requestBody = JSON.stringify({  
 problem\_id: problemId,  
 goal\_description: goalData.description,  
 target\_date: goalData.targetDate  
 });  
  
 dispatch('MAKE\_ADD\_GOAL\_REQUEST', {requestBody});  
},  
  
'ADD\_GOAL\_SUCCESS': (coeffects) => {  
 const {action, state, updateState, dispatch} = coeffects;  
  
 // Use stored problem ID to refresh  
 const problemId = state.lastAddedGoalProblemId;  
  
 // Clear loading state  
 const updatedSavingGoals = {...state.savingGoals};  
 delete updatedSavingGoals[problemId];  
  
 updateState({  
 savingGoals: updatedSavingGoals,  
 lastAddedGoalProblemId: null  
 });  
  
 // Refresh problem's goals  
 if (problemId && state.currentAssessmentId) {  
 dispatch('LOAD\_PROBLEM\_GOALS', {  
 problemId: problemId,  
 guidelineTemplateId: state.currentAssessmentId  
 });  
 }  
  
 // Show success message  
 dispatch('ADD\_SYSTEM\_MESSAGE', {  
 type: 'success',  
 message: action.payload.message || 'Goal added successfully'  
 });  
}

**Why State-Based Refresh**: Effect meta parameters are unreliable; storing context in state ensures proper refresh

## API Endpoint Reference

### Complete Endpoint List

**Assessment Management**: - POST /api/.../careiq\_api/get-assessments → GET /builder/guideline-templates - POST /api/.../careiq\_api/create-assessment → POST /builder/guideline-template - POST /api/.../careiq\_api/get-sections → GET /builder/guideline-template/{id}/sections - POST /api/.../careiq\_api/publish-assessment → PUT /builder/guideline-template/{id}/publish - POST /api/.../careiq\_api/create-version → POST /builder/guideline-template/{id}/version

**Section Management**: - POST /api/.../careiq\_api/add-section → POST /builder/guideline-template/{id}/section - POST /api/.../careiq\_api/update-section → PUT /builder/section/{id} - POST /api/.../careiq\_api/delete-section → DELETE /builder/section/{id} - POST /api/.../careiq\_api/get-section-questions → GET /builder/section/{id}/questions

**Question Management**: - POST /api/.../careiq\_api/add-question-to-section → POST /builder/section/{id}/questions - POST /api/.../careiq\_api/add-answers-to-question → POST /builder/question/{id}/answers - POST /api/.../careiq\_api/update-question → PUT /builder/question/{id} - POST /api/.../careiq\_api/delete-question → DELETE /builder/question/{id}

**Answer Management**: - POST /api/.../careiq\_api/update-answer → PUT /builder/answer/{id} - POST /api/.../careiq\_api/delete-answer → DELETE /builder/answer/{id}

**Relationship Management**: - POST /api/.../careiq\_api/answer-relationships → GET /builder/answer/{id}/relationships - POST /api/.../careiq\_api/add-branch-question → POST /builder/answer/{id}/triggered-question - POST /api/.../careiq\_api/delete-branch-question → DELETE /builder/triggered-question/{id} - POST /api/.../careiq\_api/add-guideline-relationship → POST /builder/answer/{id}/guideline - POST /api/.../careiq\_api/delete-guideline-relationship → DELETE /builder/guideline-relationship/{id}

**PGI Management**: - POST /api/.../careiq\_api/load-problem-goals → GET /builder/problem/{id}/goals - POST /api/.../careiq\_api/add-goal → POST /builder/problem/{id}/goal - POST /api/.../careiq\_api/load-goal-interventions → GET /builder/goal/{id}/interventions - POST /api/.../careiq\_api/add-intervention → POST /builder/goal/{id}/intervention

**Search**: - POST /api/.../careiq\_api/generic-typeahead → GET /builder/search

**Configuration**: - POST /api/.../careiq\_api/get-careiq-config → GET /config - POST /api/.../careiq\_api/use-case-categories → GET /builder/use-case-categories

## Best Practices

### Client-Side

**DO**: ✅ Use POST for all ServiceNow endpoints ✅ Send direct fields (no data wrapper) ✅ Handle both success and error actions ✅ Clear loading states in all handlers ✅ Show user feedback (system messages) ✅ Store context in state for multi-step operations ✅ Reload data after successful saves

**DON’T**: ❌ Use GET for ServiceNow endpoints (inconsistent) ❌ Wrap request in data: {} (ServiceNow does it) ❌ Rely on effect meta parameters for critical data ❌ Leave loading states active on error ❌ Skip user feedback on operations

### ServiceNow API

**DO**: ✅ Stash request.body.data immediately ✅ Validate required fields ✅ Use safe error handling (no e.message) ✅ Return consistent response structure ✅ Set appropriate HTTP status codes ✅ Instantiate Script Includes correctly

**DON’T**: ❌ Access request.body.data multiple times ❌ Use e.message or e.stack in catch blocks ❌ Return inconsistent response formats ❌ Skip error handling ❌ Expose sensitive data in error messages

### Script Include

**DO**: ✅ Centralize business logic ✅ Handle authentication centrally ✅ Transform data between formats ✅ Provide clear method names ✅ Return consistent structures

**DON’T**: ❌ Put business logic in REST APIs ❌ Hard-code credentials ❌ Skip error handling ❌ Return raw CareIQ responses without transformation

## Troubleshooting API Issues

### Request Not Reaching ServiceNow

**Check**: - Network tab in browser DevTools - Correct endpoint URL - HTTPS protocol - ServiceNow session active

### Request Data is Undefined

**Check**: - Request body is JSON string - ServiceNow API uses request.body.data - Data was stashed immediately

### Script Include Method Not Found

**Check**: - Correct scope: x\_1628056\_careiq.CareIQServices - Script Include is active - Method name spelling - Method exists in consolidated services

### CareIQ Platform Error

**Check**: - Authentication token valid - Correct CareIQ API URL - Request payload format matches CareIQ API spec - CareIQ platform is available

### Response Not Updating UI

**Check**: - Success action is dispatched - Success handler updates state correctly - State update triggers re-render - No JavaScript errors in console

## Summary

CareIQ Builder’s API communication: - **Three-tier architecture**: Client → ServiceNow → CareIQ - **Security-first**: No direct external API calls - **POST everywhere**: All ServiceNow endpoints use POST - **Direct fields**: Component sends unwrapped data - **Safe error handling**: No e.message access in ServiceNow - **State-based context**: Reliable typeahead and refresh patterns - **Consistent responses**: Standard success/error format

This architecture provides security, maintainability, and predictable API communication.