PassQueen
a.k.a dumpsqueen.com

Associate CertificationInsuranceSuite DeveloperMammoth Proctored Exam

Guidewire InsuranceSuite-Developer

Version Demo

Total Demo Questions: 17

Total Premium Questions: 173

Buy Premium PDF

https://passqueen.com

support@passqueen.com

QUESTION NO: 1

An insurer has a number of employees whose names are similar, but each one has a unique employee number for identification. Displaying the employee ' s name as a drop-down list in the user interface must include the employee ' s number with the employee ' s name to ensure uniqueness. For example:

John Smith 3455

William Andy 3978

John Smith 4041

How can a developer satisfy this requirement following best practices?

A. Enable Post On Change for name fields to modify how they display when the name is referenced.

B. Define an entity name that concatenates the name fields and employee number.

C. Create a setter property in a Name enhancement class.

D. Create a Displaykey that concatenates the name fields and employee number.

ANSWER: B

Explanation:

Define an entity name that concatenates the name fields and employee number. In Guidewire, an entity's name/display-name configuration provides the standard human-readable representation of an entity instance. UI controls that present entity references, including selection controls and lists, can use that representation rather than requiring each individual screen to implement its own formatting logic. Defining the employee representation centrally as a combination of the employee's name fields and unique employee number produces values such as "John Smith 3455" and "John Smith 4041," allowing users to distinguish otherwise identically named employees.

This is a best-practice solution because the identifying text is derived from the employee data model and is reusable wherever an Employee is displayed. It also keeps the employee number as part of the displayed identity without changing the underlying identifier or storing a duplicated formatted value. The configuration should use the appropriate employee name fields and the employee-number field so the value remains accurate as employee records are maintained. Guidewire's developer documentation describes data-model entities and their configuration as the basis for application behavior; see the Guidewire Documentation portal and the Guidewire Resources for product documentation access and developer materials.

QUESTION NO: 2

Which statements about Gosu package structure and naming conventions follow Guidewire best practices for customer implementations? (Choose 2)

A. Use underscores in package names (e.g., my_package).

B. Place all custom classes in a single custom package.

C. Use generic package names like com.company.

D. Include the product code (e.g., pc, cc, bc) in the package structure.

E. Group classes solely by functional area (e.g., util, entity, batch).

F. Use the customer code as the top-level package segment.

ANSWER: D F

Explanation:

Using the customer code as the top-level package segment provides a distinct namespace for implementation-specific Gosu code. For example, beginning packages with an insurer-specific identifier helps make ownership immediately clear and keeps customer extensions separate from Guidewire-delivered code, which commonly uses the `gw` namespace. This separation is especially valuable during upgrades because developers can more easily identify and review custom code without confusing it with product code.

Including the product code in the package structure further organizes code when an organization configures more than one InsuranceSuite application. A hierarchy such as `acme.pc.integration` or `acme.cc.util` communicates both the owning customer namespace and the intended application context. It supports predictable navigation, reduces accidental coupling between product-specific implementations, and leaves room for clearly scoped shared components where appropriate. Package names should remain lowercase and use dot-separated hierarchical segments, following normal Gosu and Java-style namespace conventions. Guidewire's documentation portal is available at Guidewire Documentation, and the Gosu language documentation describes packages and namespaces at Gosu Language Documentation.

QUESTION NO: 3

An insurer ran the DBCC checks against a copy of their PolicyCenter production database in a non-production environment to check for errors before promoting the code to production. Three errors with high counts were found in the category " Data update and reconciliation. " What are two best practices for resolving the errors? (Select two)

A. Identify any bad data and write a SQL script to correct the data; run the script immediately.

B. Wait to see if error counts increase; if they increase by more than 10%, fix the errors.

C. Promote the code to production and run the DBCCs again to see if the same errors are found.

D. Analyze the errors to determine the root cause and correct the code responsible for the errors.

E. Search the Knowledge Base on the Guidewire Community for solutions to the problems found.

ANSWER: D E

Explanation:

"Analyze the errors to determine the root cause and correct the code responsible for the errors" is a best practice because Data update and reconciliation DBCC findings commonly indicate that application logic, configuration, integrations, or a deployment process has produced data that does not meet PolicyCenter's expected consistency rules. Investigating the affected records and the associated check first allows the team to identify the source of the issue and prevent the same condition from continuing to create additional inconsistent data. Any remediation of existing records should follow a supported, reviewed approach after that cause is understood.

"Search the Knowledge Base on the Guidewire Community for solutions to the problems found" is also appropriate. DBCC messages can correspond to known product behavior, upgrade considerations, configuration patterns, or documented corrective procedures. The Guidewire Community provides knowledge articles and support resources that can help the team determine whether a finding has a known resolution and whether Guidewire recommends a particular data-repair or configuration process. Using those resources alongside root-cause analysis supports a controlled remediation before production promotion. See the Guidewire Community and the Guidewire Documentation portal for product documentation and support guidance.

QUESTION NO: 4

A customer needs the ability to categorize claims based on business needs. Which actions below follow best practices? (Choose two)

A. Define ClaimCategory_Ext as an extension of an existing claim Typelist.

B. Add a ClaimCategory_Ext Typekey to the Claim entity

C. Create a .ttx file for ClaimCategory_Ext in the Extensions\Typelist folder

D. Add a ' foreignkey ' to the ClaimCategory_Ext typelist that references the Claim entity

E. Name the Typelist ClaimCategory without an _Ext suffix.

F. Create a .tti file for ClaimCategory_Ext in the Extensions\Typelist folder

ANSWER: B F

Explanation:

Creating a .tti file for ClaimCategory_Ext in the Extensions\Typelist folder is the appropriate way to introduce a new custom typelist. A typelist defines the controlled set of category values available to users and business logic. Using a customer-specific name with the `_Ext` suffix clearly identifies the metadata as an extension, helping separate implementation-specific configuration from base product metadata during maintenance and upgrades.

Adding a ClaimCategory_Ext Typekey to the Claim entity connects that controlled list of values to each claim. A typekey field stores a typelist code and gives ClaimCenter a consistent, validated mechanism for assigning, querying, displaying, and reporting on a claim's category. Together, the new typelist definition and the typekey field model the requirement as configurable reference data rather than as free-form text, which supports data quality and makes category values reusable throughout rules, UI configuration, and reporting.

This approach aligns with ClaimCenter's metadata-driven data model and its use of typelists for enumerated business values. See the Guidewire Documentation portal and Guidewire's ClaimCenter product information for platform and ClaimCenter documentation resources.

QUESTION NO: 5

Automated inspections help enforce quality by identifying anomalous code and adherence to defined metrics. Which types of issues or rules are typically enforced by Guidewire Studio Inspections? Select Two

A. Detection of unaccounted-for time (Own Time) during server round trips, indicating inefficient processing loops.

B. Enforcement of naming standards for method and variable declarations across the entire Gosu configuration.

C. Measurement of the Cyclomatic Complexity metric to ensure methods do not exceed 40 statements.

D. Verification of data integrity to ensure that required columns on subtypes are correctly populated (a platform-level Database Consistency Check).

E. Identification of potential programming bugs, such as empty if or else statements or unused loop variables.

F. Detection of memory leaks caused by large, long-running bundles that were not paged correctly during batch modification.

ANSWER: B E

Explanation:

Guidewire Studio Inspections are static-analysis checks that run against configuration and Gosu source while developers work in Studio. Their purpose is to promote consistent, maintainable code and to surface likely implementation defects before the application is run or code is submitted for review. Accordingly, *Enforcement of naming standards for method and variable declarations across the entire Gosu configuration.* is a typical inspection use case. Consistent naming improves readability, discoverability, and alignment between customer configuration and the surrounding InsuranceSuite codebase.

Identification of potential programming bugs, such as empty if or else statements or unused loop variables. is also a core static-inspection function. Empty control-flow blocks and unused variables can signal unfinished logic, accidental omissions, or code that no longer has an effect. Flagging these patterns in the editor lets a developer correct them early, without requiring a server execution or a database validation process. Studio's inspection model therefore supports code hygiene and early defect prevention directly in the development workflow. Guidewire's product documentation is available through the Guidewire Documentation portal; Gosu language concepts relevant to source-level analysis are described in the Gosu documentation.

QUESTION NO: 6

Which statements describe best practices when using bundles in Gosu to save new entities/edit existing entities? (Select Two)

A. Commit changes individually for each entity.

B. Create a new bundle using gw.transaction.Transaction.runWithNewBundle().

C. Explicitly call the commit() method on the bundle outside of a managed block.

D. Add all entities to the bundle, not just those which will be edited.

E. Obtain a bundle using gw.transaction.Transaction.getCurrent().

F. Never call commit() within a runWithNewBundle() statement.

ANSWER: B F

Explanation:

Create a new bundle using gw.transaction.Transaction.runWithNewBundle(). is a best practice because it establishes an isolated, explicitly scoped database transaction for work that creates or modifies entities. The callback receives the writable bundle to use for creating entities or bringing existing entities into that bundle. This makes transaction ownership clear, avoids inadvertently modifying objects in an unrelated current transaction, and provides a reliable boundary for error handling.

Never call commit() within a runWithNewBundle() statement. is also correct because `runWithNewBundle` is a managed transaction helper. When its callback completes successfully, Guidewire manages committing the bundle; if the callback fails, the transaction is handled as unsuccessful instead. Allowing the managed API to control completion preserves its intended atomic transaction behavior and avoids manually ending the transaction from inside its own managed callback. The transaction API is designed to encapsulate this lifecycle, while the supplied bundle is the context in which entity changes are made. See the Gosu documentation at Gosu Documentation and Guidewire's developer documentation portal at Guidewire Documentation.

QUESTION NO: 7

An insurer stores the date a company was established in the company records. A business analyst identified a new requirement to calculate a company ' s years in business at the time a loss occurred. The years in business will be determined using the date established field and the claim date of loss.

The image below shows the Contact structure in the data model:



Which configuration steps will satisfy the requirement? (Select two)

A. Create a new enhancement class for the Company entity under the insurer package

B. Create a function to calculate the years In business in a Company enhancement

C. Create a setter property to calculate the years in business in the Contact enhancement

D. Create a new enhancement class for the Contact entity under the gw package

E. Create a function to calculate the years in business in a UI Helper class under the gw package

F. Create a getter property to calculate the years in business in a Company enhancement

ANSWER: F

Explanation:

Create a getter property to calculate the years in business in a Company enhancement. A Gosu enhancement is the appropriate extension mechanism for adding derived behavior to an existing entity without modifying the base entity definition. Because the established-date field belongs to the Company subtype, placing the calculated behavior on Company makes the property available wherever a Company instance is used and keeps the business logic aligned with the entity that owns the source data.

A getter property represents a value that is computed when it is requested rather than independently stored or manually maintained. The getter can use the company's date-established value and the relevant claim's date of loss to derive the completed number of years in business. Consumers such as rules, PCF expressions, and Gosu code can then read the derived value with normal property syntax. This supports a clear domain model and ensures that the displayed or evaluated result is based on the current source dates.

Guidewire uses Gosu for InsuranceSuite application logic, including entity extensions and enhancements. See the Guidewire Documentation portal and the Gosu language documentation for language concepts including properties and enhancements.

QUESTION NO: 8

An insurer plans to offer coverage for pets on homeowners policies. Whenever the covered pet Is displayed in the user interface, it should consist of the pet ' s name and breed. For example:

How can a developer satisfy this requirement following best practices?

A. Enable Post On Change for the pet name field to modify how it displays when referenced

B. Define an entity name that concatenates the pet ' s name and breed fields

C. Create a setter property in a Pet enhancement class

D. Create a display key that concatenates the pet ' s name and breed

ANSWER: B

Explanation:

Define an entity name that concatenates the pet ' s name and breed fields. An entity name is the appropriate declarative mechanism when an entity instance needs one consistent, human-readable representation throughout the InsuranceSuite user interface. The entity-name expression can combine the Name and Breed values into the required display string, such as `this.Name + " - " + this.Breed`. Widgets and UI locations that render a reference to the pet can then use that entity representation automatically, avoiding repeated formatting logic in individual PCF files or Gosu methods.

This design centralizes the presentation rule with the entity's metadata. As a result, the same pet label is used consistently in selectors, lists, detail views, and other UI contexts where the platform displays the pet entity. It also makes future changes straightforward: if the required label later includes another attribute, the developer updates the entity-name definition in one place rather than locating every screen that displays pets. This aligns with Guidewire's metadata-driven configuration approach, which keeps common entity identification and display behavior centralized and maintainable. Guidewire documentation and learning resources are available through the Guidewire Documentation portal and the Guidewire Education site.

QUESTION NO: 9

Which scenario follows best practices for user interface field-level validation?

A. Proposed changes to user passwords contain only alphanumeric characters.

B. Store different social media profile addresses, regardless of the social media site involved.

C. The city and state populate automatically whenever a US user enters their ZIP code.

D. The interest rate field is 0 whenever a down payment is greater than €5000.

ANSWER: A

Explanation:

“Proposed changes to user passwords contain only alphanumeric characters.” is an appropriate example of user interface field-level validation because the rule applies to the value in one input field and can be evaluated as the user enters or leaves that field. A field-level constraint such as an allowed-character pattern provides immediate, focused feedback at the point of data entry. It prevents a structurally invalid value from progressing through the user workflow and keeps the UI behavior clear: the password field accepts values matching its defined character constraint.

In Guidewire PCF-based interfaces, simple input constraints belong close to the input widget because they are concerned with the shape and permitted content of a single value, rather than a broader business decision. The exact password policy should be defined by the application’s security requirements; this scenario’s alphanumeric requirement is the stated policy being enforced. Implementing that policy as a field-level check gives users prompt correction guidance and supports clean, predictable data capture. Guidewire product documentation is available through the Guidewire Documentation portal. For general secure-password policy considerations, see the NIST Digital Identity Guidelines.

QUESTION NO: 10

The results of a Guidewire Profiler analysis on a web page showed a large unaccounted-for time. The developer cannot identify which block of code is taking up so much time by examining the profiler output. Which approach can help to account for the large time spent and improve reading of the profiler output?

A. Identify the PCF file name in the profiler output.

B. Print out the timestamp before and after the code blocks.

C. Surround the blocks of code with profiler tags.

D. Create a function to calculate processing time in the class.

ANSWER: C

Explanation:

Surround the blocks of code with profiler tags. Guidewire Profiler captures timing for instrumented framework operations, but elapsed time in custom Gosu processing can otherwise appear as unaccounted-for time in a profile. Adding profiler tags around suspected areas—such as loops, calculations, integration preparation, or other application logic—creates named timing entries for those blocks. The resulting profile attributes the elapsed time to the supplied tag names, making it possible to isolate the expensive work and interpret the call hierarchy and timing data more effectively.

Profiler tags are particularly valuable when narrowing a broad timing gap into smaller, meaningful measurements. A developer can place tags around progressively more specific sections of a workflow, rerun the scenario, and compare inclusive and exclusive durations in the profiler output. This keeps the investigation within the standard profiling mechanism, where the tagged code is visible in context with request processing, database activity, and other profiled operations. Once the costly area is identified, the tag can be retained for diagnostic observability where appropriate or removed after performance work is complete. Guidewire’s documentation portal is available at Guidewire Documentation; profiling and performance diagnostics are also covered in the product-specific developer documentation available through the Guidewire Community.

QUESTION NO: 11

Guidewire Home provides self-service capabilities for managing storage access permissions for InsuranceSuite. According to the training, which app in Guidewire Home is used for this purpose?

A. The Storage Access app

B. The Quality Gates app

C. The Lifecycle Manager app

D. The Build Promotion app

E. The Planets app

ANSWER: A

Explanation:

The Storage Access app is the Guidewire Home self-service application used to manage access to storage resources associated with InsuranceSuite environments. Storage-based integrations commonly need controlled read and write access to exchange files, process inbound data, produce outbound extracts, or support other operational workflows. This access must be managed through a governed process so that credentials and permissions are issued only to authorized users and integration processes.

Using the Storage Access app centralizes this activity within Guidewire Home rather than requiring routine requests to be handled manually. It supports the cloud operating model by giving appropriately authorized teams a defined location for administering storage access while maintaining security controls around sensitive insurance data. The app's purpose is specifically aligned to storage permissions, making it the appropriate choice when the task is to configure or manage access to cloud storage used by InsuranceSuite.

Guidewire Cloud provides the platform services and operational tooling that support InsuranceSuite deployments, including secure, managed cloud operations. See the Guidewire Cloud overview and Guidewire's support and documentation resources for further platform information.

QUESTION NO: 12

An insurer doing business globally wants to use a validation expression to verify that a contact ' s postal code is a real postal code for the country specified in the contact ' s address.

A developer has created a method with the signature validatePostalCode(anAddress: Address): boolean, which returns true if and only if the postal code is valid.

What would be the correct validation expression?

A. validatePostalCode(anAddress) == true

B. validatePoslalCode(anAddress) == null

C. validatePostalCode(anAddress)

D. validatePostalCode(anAddress) ? null: false

E. validatePostalCode(anAddress) ? null : "Invalid postal code"

ANSWER: E

Explanation:

`validatePostalCode(anAddress) ? null : "Invalid postal code"` is correct because a Guidewire validation expression signals a successful validation by returning `null`. When the address passes `validatePostalCode`, the conditional expression therefore evaluates to `null`, allowing the contact address to be saved. When the method returns `false`, the expression returns a non-null error message. That message is the validation failure result and can be shown to a user or otherwise reported by the validation framework.

The conditional also correctly adapts the helper method's Boolean contract to the validation-expression contract. The helper answers whether the postal code is valid, while the validation expression must produce either no error result or an error message. Returning a descriptive string, rather than merely a Boolean value, provides a usable validation message and

preserves the expected result type for the validation mechanism. In a production implementation, the literal message would normally be replaced with a display key so it can be localized for users in different countries. Guidewire's documentation portal is available at Guidewire Documentation; its developer resources and training information are available through Guidewire Education.

QUESTION NO: 13

A developer needs to create an interface in a customer package to handle banking services. An implementation has been identified for online credit union services. Which interface and implementation classes adhere to the naming conventions?

A. BankService_Ext.gs, CreditUnionBankService.gs

B. IBankService.gs, CreditUnionBankService.gs

C. BankService_Ext.gs, CreditUnionBankServiceImpl.gs

D. BankService.gs, CreditUnionBankService.gs

ANSWER: B

Explanation:

`IBankService.gs, CreditUnionBankService.gs` follows the Guidewire/Gosu naming pattern for a customer-defined service abstraction and its concrete implementation. The `I` prefix makes `IBankService` immediately recognizable as an interface, clearly expressing that it defines a contract rather than a usable service instance. This is especially helpful when the interface can have multiple implementations, such as a credit-union integration, a retail-bank integration, or a test double used in automated tests.

`CreditUnionBankService` is an appropriately descriptive concrete-class name. It identifies both the provider or business context, Credit Union, and the responsibility, Bank Service, without adding a redundant implementation suffix. The class name therefore reads naturally where it is instantiated, injected, or registered, while its declared type can remain `IBankService` to preserve loose coupling. The `.gs` extension is the standard source-file extension for Gosu types. Guidewire development uses Gosu for application customization; see the Guidewire Documentation portal and the Gosu language documentation for platform and language reference material.

QUESTION NO: 14

A query is known to return 500,000 rows. Which two are recommended to process all 500,000 rows efficiently? (Select two)

A. Use Google Iterables

B. Use setPageSize()

C. Use a batch process for large result sets

D. Sort the result by entity name

E. Chunk the results into page sets

ANSWER: B E

Explanation:

`Use setPageSize()` is appropriate because a query with hundreds of thousands of rows must be consumed incrementally rather than materialized as one large in-memory result. Setting a page size instructs the query result processing mechanism to retrieve and iterate through a bounded number of rows at a time. This constrains the working-memory footprint while still allowing the application to visit every matching entity. The page size should be selected based on the processing work performed per entity and the available application-server resources.

`Chunk the results into page sets` applies the same essential large-result-set pattern at the processing level. Each page is handled and released before the next page is requested, avoiding the unnecessary retention of all 500,000 entity instances in a collection. This approach supports predictable memory usage, permits progress-oriented processing, and is suitable whether the work updates entities, creates related records, or performs calculations. In Guidewire applications, query design and result iteration should favor bounded retrieval for high-volume operations. Consult the official Guidewire Documentation portal for the product-version-specific Query API reference and the Guidewire Resources site for platform development guidance.

QUESTION NO: 15

The business wants to create a new popup in BillingCenter that displays a single customer invoicing inquiry. The popup will have the inquiry date, inquiry contact, and the description of the inquiry. Which configurations follow best practices to make this page editable? (Choose Two)

A. Add a Boolean variable named editable_Ext to the Variables tab and set its initial value to true.

B. Set the Page ' s startInEditMode property to true if it should initially be editable.

C. Set the Detail View panel ' s readOnly property to false.

D. Be sure that the ListView container widget has its editable property set to true.

E. Ensure that Input widgets are used for fields requiring data entry, and that their editable property is set to true.

F. Add an InputSet widget within the detail view and set its canEdit property to true.

ANSWER: B E

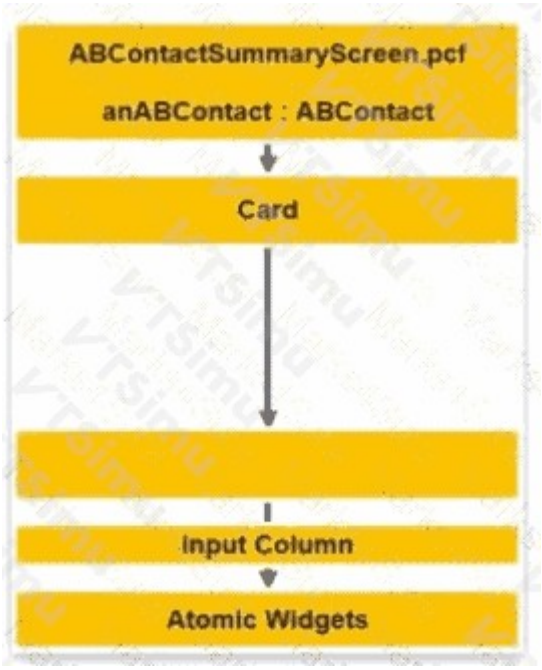
Explanation:

Setting the Page's `startInEditMode` property to `true` is appropriate when the popup is intended to open directly for data entry. This makes the initial page state explicit and avoids requiring a user to switch from a view state into an edit state before entering the inquiry date, contact, and description. It is particularly suitable for a popup that represents an inquiry being created or updated as a single business object.

Using Input widgets for values that users must enter or change, with each applicable widget's `editable` property set to `true`, provides the field-level editing behavior needed by the popup. PCF uses declarative widget configuration, so editable input controls should be bound to the inquiry properties that the user is authorized to modify. This approach keeps the configuration clear: the page starts in an editing state and the controls that accept the inquiry data are configured to accept input. Guidewire’s product documentation portal provides the applicable BillingCenter and PCF configuration references at Guidewire Documentation. Additional platform and developer resources are available from Guidewire Resources.

QUESTION NO: 16

Given the image:



Which container type must be added between Card and Input Column?

A. Detail View PCF File

B. Detail View

C. List View

D. Input Set

ANSWER: B

Explanation:

Detail View is the required intermediate container. In PCF, a Card provides the content area for a card-based UI layout, while a Detail View panel provides the detail-editing layout that can contain input-oriented widgets. The Detail View establishes the appropriate structural context for arranging editable fields and their labels. Within that detail layout, an Input Column is used to organize field widgets vertically into a column.

Accordingly, the intended hierarchy is Card, then Detail View, then Input Column. This composition lets the PCF renderer apply the detail-view layout rules before it lays out the individual input columns and the fields placed within them. It also reflects the standard Guidewire Studio approach of selecting a parent layout container first, then adding the input-column containers used to organize the page's editable content. Keeping this hierarchy makes the card content valid and ensures that input fields inherit the expected detail-panel spacing, labels, and alignment behavior.

Guidewire's product documentation portal is available at Guidewire Documentation, and Guidewire's developer resources are available at Guidewire Developer.

QUESTION NO: 17

When creating an entity enhancement in Gosu, which of the following practices are recommended? (Choose 2)

A. Use the suffix _Ext for new properties added to base application entities.

B. Use a noun for most properties, but use an adjective for boolean properties.

C. Getters do not need to be null safe.

D. Use the suffix _Ext for new methods added to custom entities.

E. An enhancement to a subtype/subclass will need to be added to each child subtype/subclass as enhancements are not automatically inherited.

F. Ensure that the enhancement file is placed in the same package as the enhanced type.

ANSWER: A F

Explanation:

Use the suffix `_Ext` for new properties added to base application entities. This naming convention makes customization boundaries explicit and reduces the risk that a future Guidewire product upgrade introduces a base property with the same name. An enhancement can add computed properties and behavior to an existing entity without modifying Guidewire source, so clear extension-oriented names help developers distinguish product functionality from implementation-specific customization. The same convention also improves maintainability when configurations are reviewed, merged, or upgraded.

Ensure that the enhancement file is placed in the same package as the enhanced type. Keeping an entity enhancement with its target type follows Guidewire's organization convention and makes the relationship clear to developers and tooling. It avoids scattering enhancements across unrelated namespaces, simplifies discovery of custom behavior, and helps ensure that the enhancement is maintained alongside the entity API it extends. Gosu enhancements are the language mechanism used to add methods and properties to an existing type; see the Gosu language documentation. For Guidewire implementation and configuration resources, consult the Guidewire Documentation portal.