### **What is Gosu?**

Gosu is Guidewire’s open-source, publicly available programming language. Gosu has elements of both procedural and object-oriented programming languages and is similar to JavaScript and Java.

### **Why Gosu?**

1. There was a desire to have a single syntax that could be used to work with all of the elements relevant to Guidewire products (such as entities, display keys, classes, class enhancements, Java classes, permissions, and script parameters) even though these items have fundamentally distinct internal implementations.
2. There was a desire to have code auto-complete features in Studio. This is possible only with a statically typed language.

### **Gosu Statements:**

* var variableName : datatype = initialValue
  + Use var keyword to declare variable
  + Use : to specify datatype
  + Use = to assign initial value
* No terminator required – GW recommends NOT to use semicolons(terminator)
* Import packages is done via uses operator

#### **Gosu Primitive types:**

* int, char, byte, short, long, float, double, Boolean
* Primitive types cannot hold null value

### **Script Parameters:**

* Script parameters act as a global variables where the variable value may change over time and therefore should not be hard-coded

#### **Steps needed to create script parameter:**

1. open ScriptParameters.xml
2. Create a parameter in XML
3. Deploy the script parameter
4. Reference the script parameter

### **Gosu Classes:**

* **Gosu classes** are similar to classes in other object-oriented language, such as Java

#### **Steps needed to follow while creating a Gosu class:**

1. Create a package
2. Create a class (Optional: extend super class or implement interface)
3. Optional Constructor, properties, methods (if extend super class or implement interface, might be required to override methods from base)
4. Loggers
5. Logging records information about system behavior that is required for diagnostics, trouble-shooting, and failure recovery. Log files contain various types of application messages saved in a separate files. After a log file has been written, you can view a log file. Guidewire logging uses slf4j log severity levels: Trace, Debug, Info, Warn, Error. Each level is considered more severe than the previous.
6. The logger is of the type org.slf4j.Logger and uses a static logger category for configuration.
7. Best practice to record information is using loggers instead of print() statements.
8. Optional Exception handling blocks
9. Optional Annotation: Annotations are a simple syntax to add metadata or a behavior to a Gosu class.
10. Optional Blocks: Block is an unnamed method that is typically created as an input parameter for another method

**Syntax:** (\arguments -> logicToExecute)

## **Gosu Business Rules:**

* A **business rule** performs an action based on a condition for a given entity when a specific event occurs to an instance of that entity
* A **rule set** is a type of Gosu class that represents a collection of rules that are attached to the same entity and share common triggers
* **Examples of business rules:**
  + - 1. Presetup
      2. Segmentation
      3. Workplan
      4. postsetup
      5. assignment
      6. preupdate
      7. validation
      8. postupdate

## **Gosu Enhancements:**

* A **Gosu enhancement** is code that enhances the functionality of an existing type
* Enhancement is tied to specific entity and this is used to add new properties to an entity.

#### **Steps needed to follow while creating a Gosu Enhancement:**

1. [Optional] Create a new package
2. Create a new enhancement file
3. Code getter properties, setter properties, methods
4. Deploy the enhancement

## **Gosu Queries:**

### **Why Gosu query?**

To retrieve data from database tables (entities), use gosu query object(s).

Specifies query criteria (Which entity and restrictions to be applied)

### **Steps to execute a query:**

1. Create the query object using the make() method
2. Apply restriction for query object
3. Create the results object using the select() method
4. Process the results of the query as needed

## **Bundles:**

### **What is a bundle?**

* A **bundle** is a set of in-memory entity instances that represent rows in the database
* In the Guidewire paradigm for integration and configuration, the term "entity" can express a duality of both being a class and a database table.
* Gosu represents a bundle with the class gw.transaction.Bundle. The Bundle class is a Guidewire class that implements the entity BundleProvider.
* Saving all entities in the bundle to the database means committing the data to database

### **Types of bundles:**

1. Read-only bundle: A read-only bundle is a bundle that contains entities retrieved from the database. This includes both entities returned as the result of a query, and entities referenced by the foreign key of a related entity when the related entity is in a read-only bundle. The entities in a read-only bundle cannot be modified.
2. Writable bundle: You must copy objects from read-only bundle to a writable bundle to change objects or delete objects.

Copying entity from read-only to writable bundle:

**origEntity = writableBundle.add(origEntity)**

### **Bundle context:**

1. Current Bundle: A current bundle that contains objects available to the current code context. The current context includes, but is not limited to, the user interface and plugins. The Guidewire application automatically creates the current bundle in order to create, read, update, and delete data. A current bundle is either read-only or writable depending on the context. Integration developers can reference entities in the current bundle using the API: gw.api.transaction.Transaction.getCurrent().

bundle.commit() statement is required to commit the current bundle.

1. New Bundle: A new bundle is a bundle created explicitly by integration code.

#### **Ways to create new bundle:**

1. **gw.transaction.Transaction.  
   runWithNewBundle(\ *newBundle* -> {  
    *CodeBlock* } )**
2. **gw.transaction.Transaction.  
   runWithNewBundle(\ *newBundle* -> {  
    *CodeBlock* } *, user* )**

**runWithNewBundle()** commits the bundle at end of block by design

1. **gw.transaction.Transaction.newBundle()**

needs explicit commit statement.

* Copying entity into a new bundle:
  + **origEntity = newBundle.add(origEntity)**

### **Creating new entity using bundle:**

**gw.transaction.Transaction.runWithNewBundle(\newBundle -> {   
 var newEntity = new entity()   
 newEntity.Field1 = "value"**

**}, "user")**

### **Original values:**

* Prior to the commit, Gosu retains a copy of each entity read from the database
* **entityName.Changed**
  + Returns true if any property on entity has changed
* **entityName.isFieldChanged("FieldName")**
  + Returns true if property has changed since it was read from database
* **entityName.getOriginalValue("FieldName")**
  + Returns value read from database

## **Gunit Tests:**

* **GUnit** is used to configure and run repeatable tests of your Gosu code in a similar fashion as JUnit works with Java code.

**Steps to create a Gunit Test:**

1. Optionally create a new test class
2. Create a test method
3. Set up test data
4. Call the method being tested
5. Verify the results

**create a new test class:** Right click on gtest -> New -> Gosu Class(Class name should end with ‘Test’) and need to extend gw.TestHarness.TestBase class

**Create a test method:** Method name should start with test followed by actual method name

**Set up test data:**

## **Gosu Entity Builders:**

* An **entity builder** is a Gosu class used to create Guidewire objects and set the properties on those objects
* Entity builders are extensions of the platform class gw.api.databuilders.DataBuilder
* Can be used to create data objects for use in Gunit testing
* Do not use in production environments - Entity builders require Internal Tools to be enabled. Internal Tools should not be enabled in a production environment

**Call the method being tested:** calling the actual method

**Verify the results:** use assert statements to verify the results