**General-Purpose Language Documentation**

**Part A. Language Basics  
1) Declaration of Data Types**

**> This GPL works with pre-declaration of types. Is possible use the variables previously declared in the type declaration layer. See Type Declaration to more information.**Examples:   
value1: int  
name: string

**> Primitive types**  
**Syntax:** (name of variable): (type name created in GPL)  
Examples:   
value1: int  
name: string

**> Reference types  
a) Class (Instance)  
Syntax:** (name of object): (class name declared)  
Examples:   
obj1: Product  
name: string

**b) Array  
- Syntax**: name of list: type name created in GPL (type) or a class name declared  
Examples:   
listProduct: listObj<Product>  
listNumbers: listNumb<int>  
  
**- Assignment**: It is possible attribute value to list through a variable, for sample:  
v1: int  
v1 := 2  
listNumb.add(v1)   
Obs: It is impossible attribute the value directly to the list

**- Get value from List:**It is possible to take a value through a variable or the value of its position:  
Thus: listNumb[**2**] **or** listNumb[**variable int**]  
Example:  
z: int  
z:= 0  
usingValue: string  
**While** z < listNumb.size {  
 usingValue:= listNumb[z]  
}

**Class declaration  
> Classes can be declared using the reserved word “class”**  
Examples:  
**class** Product{  
}

**> Classes can be extended by the reserved word “extends”**Examples:  
class Customer **extends** Person{  
}

**2) Operators**

**a) Conditional Operators   
- &&** represents the **AND;  
- ||** represents the **OR.**Examples:   
**If** z>=10 **&&** z<20 {

e:= 1+1

}

**If** z==10 **||** ==20 {

e:= 2+2

}  
  
**b) Relational Operators:**  
 **==** represents **equal to;  
 !=** represents **not equals to;**

**>** represents **greater than;**  
 **>=** represents **greater than or equal to;**  
 **<** represents **less than; e**  
 **<=** represents **less than or equal to.**

**c) Arithmetic Operators  
 +** Additive operator (also used for String concatenation) **-** Subtraction

**\*** Multiplication

**/** Division  
 **%** Remainder 

**d) Unary Operators  
++** Increment operator; increments a value by 1 **--** Decrement operator; decrements a value by 1

**e) Assignment Operator  
:=** Simple assignment operator  
Examples:   
x: int  
a: int  
b: int  
a **:=** 5  
b **:=** 9  
x **:=** 2 + 3 \* (a/b)

**3) Control Flow Statements  
> This GPL works with conditional, looping and the branching statements  
  
a) Condicional Rules: If, ElseIf and Else.** Always the first letter starting with uppercaseSyntax:   
**If** (expression) {   
 statement(s)  
}**ElseIf** (expression) {  
 statement(s)  
}**Else**{  
 statement(s)  
}  
Examples:   
**If** z<10 {

e:= 1+1

} **ElseIf** z>=10 && z<20{

e:= 2+2

} **Else** {

e:= 3+3

}  
 **b) Looping statements:   
Looping Rules:** Always the first letter starting with uppercase**.  
I –) While  
Syntax**:   
While expression {

statement(s)

}  
**Example**:   
**While** z < 10 {

e := 5+2

}  
  
**II –) Do-While  
Syntax**:   
Do {

statement(s)

}While expression  
**Example**:   
**Do{**

e := 5+9

**}While** z<10

**II –) For  
Syntax**:   
for initialization, expression, increment {

statement(s)

}  
 **Example**:   
z: int

**For** z=1,z<5,z++ {

e:=z

} **c) Branching Statements  
I -) Break:** You can also use break to terminate a For, While, or Do-While loop

**Example**:   
z: int

**For** z=1,z<5,z++ {

**If**(z==2){

**break**

}e:=z+1   
} **II -) Continue:** works opposite to break in For, While, or Do-While loop

**Example**:   
z: int

**For** z=1,z<5,z++ {

**If**(z==2){

**continue**

}e:=z+1   
}

**III -) Return:** control flow returns a value to where the method was invoked  
**Example**:   
**operation** addValues(v1: int, v2: int): int{

total: int

total := v1 + v2

**return** total

}

**4) Classes and Objects  
>** Classes are declared using the reserved word **class.** Example:  
**class** Product{

id: int

name: string

description: string  
 **operation** Product(startName: string){ //constructor

name := startName

}

}  
  
**class** Bike **extends** Product{

gear: int

speed: int

**operation** Bike(startGear: int, startSpeed: int){ //constructor

gear := startGear

speed := startSpeed

}

}

**>** For each variable declared within the class Get and Set methods are automatically generated. The extension of a class is done with the reserved word **extends.**

**a) Defining Methods**

**Syntax**:   
**operation** methodName (parameter list): method type {

statement(s)

return type case in different case of type method void

}   
  
**Example**:   
**operation** AuthenticationUser(cli: User): list<string>{

objCl: User

accessData: list<string>

listObjCli: list<User>

...

**return** accessData

}

**b) Creating Objects   
Syntax**:   
object name: Class  
  
**Example**:   
objProd: Product  
  
**c) Using Objects   
Syntax**:   
object name.(methods orvariable)  
**Example**:   
objProd.insertProduct(objProduct)  
objProd.name := “Ball”

**Part B. Multiplataform Programming Concepts**

**1) Language Paradigm.  
-** This language uses the **Object-Oriented (OO)** paradigm for better behavior.  
- It is important standardization of class names as follows:

|  |  |  |
| --- | --- | --- |
| **Class type** | **Syntax** | **Samples** |
| POJO Class | Entity name | Product, Person, Car… |
| DAO Class | Entity name + “DAO” | PoductDAO, PersonDAO, CarDAO… |

**2) Language Architecture.**  
This language uses the **Model View Control (MVC)** architecture for better organization.  
  
**3) Approach structure.**The approach is structured in 4 layers:  
A - **Type Declaration;**B - **Global functions declarations;**  
C - **Deploy Model;**  
D - **CrossPlatform system model; e**  
E - **Platform Models.**  
  
**A - Type Declaration.  
In this layer is possible make the pre-declaration of types that will be used in your project.  
> Creation of Variables** **Syntax:** (datatype) (variable type)  
Examples:   
**datatype** int

**datatype** string

**datatype** float

**datatype** double

**datatype** bool

**datatype** dateTime

**datatype** void

**datatype** list<E>

**datatype** linkedHashMap<E, T>

**datatype** img  
**Obs:** Lists can be generated using generic types, so lists of any created type can be created.  
  
**B - Global functions.**Global functions are methods that can be customized in platform models to generate code that fits for different languages. The goal is to use a unique system modeling method that can be adapted to target platforms. Global functions can be customized in real time, generalizing their use to different contexts.  
 **There are two type of global functions:  
- Global:** Used to generate custom methods for different languages, defined in Platform Models. Used to generate custom methods in real time for different languages, defined in platform models. As examples, we can cite data persistence methods, methods to get data from some database, methods with platform-specific implementation details (example: TableView for iOS and Adapter´s for Android or LocationGPS).

**- Global functions declarations.**  
  
**Syntax:** global <generic types> function name (params): return type  
Examples:   
**global** <E> InsertObject(e: <E>): string   
**global** <E> DeleteObject(e: <E>): string   
**global** <E> SelectObject(ord: string): list<E>   
**global** <E, T> getViewConstruction(): <E>   
**global** <T> tableView(): <T>   
  
**- Global Details:** It is used to add details of platform-specific implementations within the classes. Standard methods or required functions for a given platform. As an example connection syntaxes.  
**Syntax:** globalDetails <generic types> function name  
Examples:   
**globalDetails** <E> DBConnectionSQLite

**globalDetails** <E> Adapter

**globalDetails** <E, T> ControllerDetails

**globalDetails** <E, T> ControllerDetailsGrid

**- Global functions implementation.  
Syntax:** implementsGlobal <generic types, ...> function name (params): return type {  
'''   
'''   
}  
The three apostrophes code (''') have the function of replicating the code placed between them in the generated code. This is done to reuse the implementation of the global function in the generated codes, since it was done using a specific language (C#, Java or Swift for example). In the code implemented between the apostrophes, Velocity programming can also be used to generate changes in code generation time.   
Example:  
1. **implementsGlobal** <E> DeleteObject(e: <E>, context: Context): string{  
2. ''' string result;   
3. Connection db = Connection.getInstance(c.getApplicationContext());   
4. String query = "DELETE FROM $E.name WHERE id=" + e.getId();   
5. try{  
6. db.getWritableDatabase().execSQL(query);   
7. db.close();   
8. return="true";  
9. }catch(Exception e){   
10. return="false";  
11. }  
12. return result;   
13. '''   
14. }  
Note that the generic parameter E is used to generalize the SQL command ($ E.name) on line 4. In this case, this function can be used by any entity, just by passing the correct entity in the function.  
  
**- Global functions call in GPL  
Syntax:** usesGlobal <generic types, ...> function name (params): return typegeneric types,

**Example:**  
usesGlobal <OrderFinalized> DeleteObject(e: OrderFinalized): string

Note that the concrete parameter OrderFinalized is passed in place of the generic parameter E in the function call.

**B.1- Examples of global functions available in the approach:**  
**B.1.1-GRID System:**Grids are used to present data in an organized way on difference device screens.  
**Obs**: In this example grids are generated using the following structures:  
 - Swift: tableView  
 - Java Android: Android’s AdapterView with ListView  
- C#: prepare data to send View layer

To generate device-specific functions, global classes were built with automatic runtime adaptations in platform models. The global class created was: **ControllerDetailsGrid.  
  
ControllerDetailsGrid:** is used to generate code for the grid environment on Android and iOS  
- Android: use onCreate and calls the adapter to build the data grid. Puts the details of class that calls the Adapter code  
- iOS: puts the details of class that generate the tableView code  
- In Web has no function.  
  
**I) Declaration:   
globalDetails** <E, D, L, M> ControllerDetailsGrid

//E-Entity D-Data L-List M-Method

**Definition**:  
E- Main entity of the method. Used toGPS define element name of generated code.

D- Data Used to popular Grid

L- Type of Lists used to carry data. By convention, put the name of the class + the reserved word “List”. Used to get the class name too.

M- Method responsible for grid business rulesGPS  
**Uses (IMPORTANT)**:  
- E and L is used for Web. This platform can send multiple lists to view.  
- E, D and L is used for android and iOS. This platform can send one(1) list to View.

**II) Syntax of call the function in the layer Cross-platfom System Model:   
Syntax: usesGlobalDetails** <Entity1, Entity2, Entity3, Entity4> ControllerDetailsGrid  
Entity1: Main Type;

Entity2: Data used to populate the grid;  
  
Entity3: List of object list used to send to grid; In Case of HashMap structured, should put in the list class the LinkedHashMap type to the list. Used to get the class name too.  
  
Entity4: Method name, return type and params of method that do anything in the start method;  
 **Example: usesGlobalDetails** <Product, ProductAdapterConfData, ProductControllerList> ControllerDetailsGrid

**Obs: it** is important create a method to populate the list to send the data grid system. The default list name is: **dataList + TypeList** and the default method name is defined in first param in Entity4.   
**Obs2:** The lists declared in Entity3 does not to be declared in GPL code, because it is declared automatically with global list.  
**Example of Entity3(L):**

**class** CartControllerList{

Order: string

}  
  
**Example of Entity4(M):  
class** CartConfMethod{

populateList: string //Method name

void: string //Method return type

sessionId: string //Parameters of method

idProd: int //Parameters of method

quant: int //Parameters of method  
 …

}  
  
I**t** isimportant to respect these names to work fine**.** That list is created automatically in both platform: Android and iOS.  
Example:

**operation** populateList(): void{

objProd: ProductDAO

ord: string

ord := "Order by name "

dataListProduct := **global** objProd.SelectObject(ord) }

**III) Implementation of function in Plataform Model:  
implementsGlobalDetails** <E, D, L, M> ControllerDetailsGrid{

'''  
 // specific code  
 '''

}

**Name Definition:  
iOS:**-Name of dataTableView: **dataTableView  
-**Name of dataTable Cell: **Main Type + CustomTableViewCell. Ex: ProductCustomTableViewCell**  
-Segue name: **showAnyViewBySegue**  
-Name of ViewController that receives the data: **AnyViewController**-Function to populate list: **populateList()**-List name is: **dataList + TypeList**

**Android:**-Function to populate list: **populateList()**-List name is: **dataList + TypeList**

**Web:**

-Function to populate list: **populateList()**-List name is: **dataList + TypeList**

**Android Adapter Generate:**

The adapter generate is create by the class Adapter using the global method: Adapter and getViewConstruction  
**Syntax:**   
class Type+Adapter extends ArrayAdapter{ }  
  
  
**Example:**   
**class** UserAdapter **extends** ArrayAdapter{ //User -> customer

**usesGlobalDetails** <Entity1> Adapter

**usesGlobal** <Entity1, Entity2> getViewConstruction(): User{}

}

**B.1.2-Database Connection:**Connections are used to perform access to local or remote databases.  
To attend the connection in different scenarios the global function DBConnection was created.  
That function was created in 3 steps:   
**I) Declaration of global function:   
globalDetails** <E> DBConnection  
  
**II) Syntax of call the function in the layer Crossplatfom System Model:   
class** Connection{

**usesGlobalDetails** <DBConf> DBConnection

}  
  
**Obs:** The class DBConf is used to put the configuration data of all connections implements. Thus, the function DBConnection implemented in Platform Model can be generic.   
  
Example of DBConf:  
**class** DBConf {

ecommerce: string //database name

db\_rest\_user: string //user name

restCommerce2019: string //password

MYSQL5011: string //server name

Smarterasp: string //server name

net: string //server name

}

**III) Implementation of function in Plataform Model:**Each platform implements the DBConnection accordingly with the language syntax using the configuration data provided by DBConf class **implementsGlobalDetails** <E> DBConnection{

'''  
 // specific code  
 '''

}

**Example of implemented function DBConnection to C# language:**  
**implementsGlobalDetails** <E> DBConnection{

'''

private MySqlConnection Conn;

public MySqlConnection OpenConnection(){

#set($server=$E.attributes.get(3).name + "." + $E.attributes.get(4).name + "." + $E.attributes.get(5).name)

Conn = new MySqlConnection("server=$server;database=$E.attributes.get(0).name;uid=$E.attributes.get(1).name;pwd=$E.attributes.get(2).name");

Conn.Open();

return Conn;

}

public void CloseConnection(){

Conn.Close();

}

'''

}  
  
Note that the attributes of generic type pass with param is used to get the data and construct the correct connection syntax. In this case is used the velocity language to generate code in real time.

**C – Platform Model**

**Syntax:**  
- call the **global class**: usesGlobal  
- pass the **generic type**: <Entity1, Entity2>   
- put the function name: **onViewCreated**(): void{  
**Example:  
  
Syntax:**- call the **global class**: usesGlobalDetails  
- pass the **generic type**: <Entity1, Entity2>  
- put the function name: ControllerDetailsGrid

**C.1- Global functions implemented to crossplatform system. Documentation.**

**C.1.1- Model  
a) DBConnection (already explained – 3-B.1.2)**

**C.1.2- Controller  
a) ControllerDetailsGrid (already explained in – 3-B.1.1)  
b) ControllerDetails:** Implements basic control class details for each device  
**c) onCreated:**

**d) onViewCreated:** is used to generate code for Java and C#.   
- In Java is generated a call to the Adapter, passing data to a grid construct.  
- In C# is generated a code to populate the data list, used to send data to the view layer.  
- In Swift has no function.  
And can be used through 3 steps:  
  
**I) Declaration:   
global** <E, T> onViewCreated(): <E>  
  
**II) Syntax of call the function in the layer Crossplatfom System Model:   
usesGlobal** <Product, ProductControllerConf> onViewCreated(): Product{}

**III) Implementation of function in Plataform Model:  
implementsGlobal** <E, T> onViewCreated(view: View, savedInstanceState: Bundle): void{

'''  
 // specific code  
 ''' }

**e) Adapter for Android:**Is used to organize the data in a grid system in Android system.  
**I) Declaration:   
globalDetails** <E> Adapter  
Used for put the default code in Adapter  
  
**global** <E, T> getView(): <E>   
Used for buid the structure responsible in send the data to layout (View)  
  
 **II) Syntax of call the function in the layer Crossplatfom System Model:**

**Rules:**  
a) Name the class: Main class + Adapter. First letter in capital letters  
b) In this case extend the BaseAdapter class  
c) Call the Global Details Adapter and pass the main class  
d) Call the Global function getView and pass the main class and the configuration class. The configuration class contains the list of data to show in the View

**Example:  
class** OrderAdapter **extends** BaseAdapter{ //Adapter. User -> customer

**usesGlobalDetails** <Order> Adapter

**usesGlobal** <Order, CartAdapterConf> getView(): View{}

}

**III) Implementation of function in Plataform Model:**Contains the main details of implementation the Adapter  
**Rules:**a) The View layer that organize the adapter layout should call: main class + \_item. Ex: product\_item;  
b) The itens of adapter layout should call: item\_ + name of data. Ex: item\_name, item\_price, etc; **implementsGlobal** <E, T> getView(position: int, view: View, parent: ViewGroup): View{

'''

'''

}

**implementsGlobalDetails** <E> Adapter{

'''  
 '''

}

**C.1.2- GPSPosition**It is used to get the current position of mobile users. It is possible uses the following data of physical position: latitude, longitude, provider, accuracy, altitude, among others available in LocationServices API.

Obs: It is important to create a string list called **dataRealLocationList** to get the location data that follows the order placed in the **GPSPositionGetDataLocation** class.

**I) Declaration:   
globalDetails** <D> ControllerGPS

This global method adds implementation details to obtain the user's current position, as well as requesting permission to use it.  
**Defination**:  
D- Entity used to pass the data of position that will be used.

**global** <E> getLocation(): list<string>

Used to populate the **dataRealLocationList** with the location data set in **GPSPositionGetDataLocation** class  
  
**II) Syntax of call the function in the layer Crossplatfom System Model:   
usesGlobalDetails** <GPSPositionGetDataLocation> ControllerGPS

**usesGlobal** <GPSPositionGetDataLocation> getLocation(): list<string>

Example of configuration class:

**class** GPSPositionGetDataLocation{

latitude: string

longitude: string

//provider: Return the name of provider that generated this fix

//accuracy: Return accuracy in meters

//altitude: Return the altitude in meters if available

}

**III) Implementation of function in Plataform Model:  
implementsGlobalDetails** <D> ControllerGPS{

'''  
 // specific code  
 ''' }

**implementsGlobal** <E> getLocation(locValue: CLLocationCoordinate2D, data: list<string>): list<String>{

'''

// specific code  
 ''' }

**Obs2:**

To use the data of location it is necessary to create a method to manipulate the string list **dataRealLocationList** with the name **useData()**. This method will be called in the inicial method of each device.

**Example of GPS use:  
class** GPSPosition{

dataRealLocationList: list<string>

**usesGlobalDetails** <GPSPositionGetDataLocation> ControllerGPS

**usesGlobal** <GPSPositionGetDataLocation> getLocation(): list<string>

**operation** useData(): void{

x: int

data: string

**For** x=0,x<dataRealLocationList.size,x++ {

data := dataRealLocationList.get(x)

}

}

}

**class** GPSPositionGetDataLocation{

latitude: string

longitude: string

}