

Modernisation interfaces Reflex

Table des matières

[1. Existant 3](#_Toc167185162)

[1.1. Principe de fonctionnement 3](#_Toc167185163)

[1.2. Description de l’outil 3](#_Toc167185164)

[2. Evolution 4](#_Toc167185165)

[2.1. Ecran général 4](#_Toc167185166)

[2.2. Ecran de paramétrage BD 5](#_Toc167185167)

[2.3. Ecran de paramétrage Général 6](#_Toc167185168)

[2.4. Ecran de sélection du contexte de reprise 7](#_Toc167185169)

[2.5. Ecran de paramétrage des correspondances 7](#_Toc167185170)

[2.6. Ecran de paramétrage technique 9](#_Toc167185171)

[2.7. Ecran de paramétrage de chaque interface 10](#_Toc167185172)

[2.8. Lancement du programme de génération du fichier d’interface 11](#_Toc167185173)

[3. Améliorations 13](#_Toc167185174)

[3.1. Interface 17 13](#_Toc167185175)

[3.2. Requêtes utilisées 13](#_Toc167185176)

[3.3. Pouvoir appeler un programme de génération d’interface spécifique 13](#_Toc167185177)

[4. Annexes techniques 13](#_Toc167185178)

[4.1. Extrait de source pour la sélection des dépôts activités 13](#_Toc167185179)

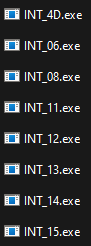
# Existant

## Principe de fonctionnement

Dans le process de migration, nous sommes amenés à reprendre des données depuis un environnement CS, 400 ou même web vers le nouvel environnement web. Dans certains cas nous reprenons des données via des interfaces (EDI) Reflex standards générées par un de nos outils.

## Description de l’outil

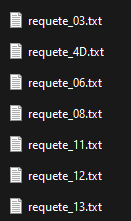
L’outil de génération d’interface Reflex fait partie de la suite Odin, il se compose de programme standalone lançable en mode console :



Il existe donc un programme par interface Reflex.  
Ces programmes utilisent des fichiers de paramétrages communs :   
corresp.json qui permet de gérer les correspondances de données en cas de changement à la volée.  
ex : changement d’un code dépôt entre l’ancien environnement et le nouveau

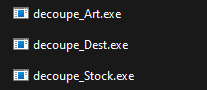
Param.xml qui permet de configurer les paramètres de connexion à la base de données et des paramètres communs

Et un fichier de paramétrage par programme qui correspond aux queues de requête des programmes



L’appel de chaque programme va générer un fichier texte contenant l’interface au format Reflex standard.

Dans le cas où le fichier généré est trop gros, il existe un outil dédié pour découper les fichiers :



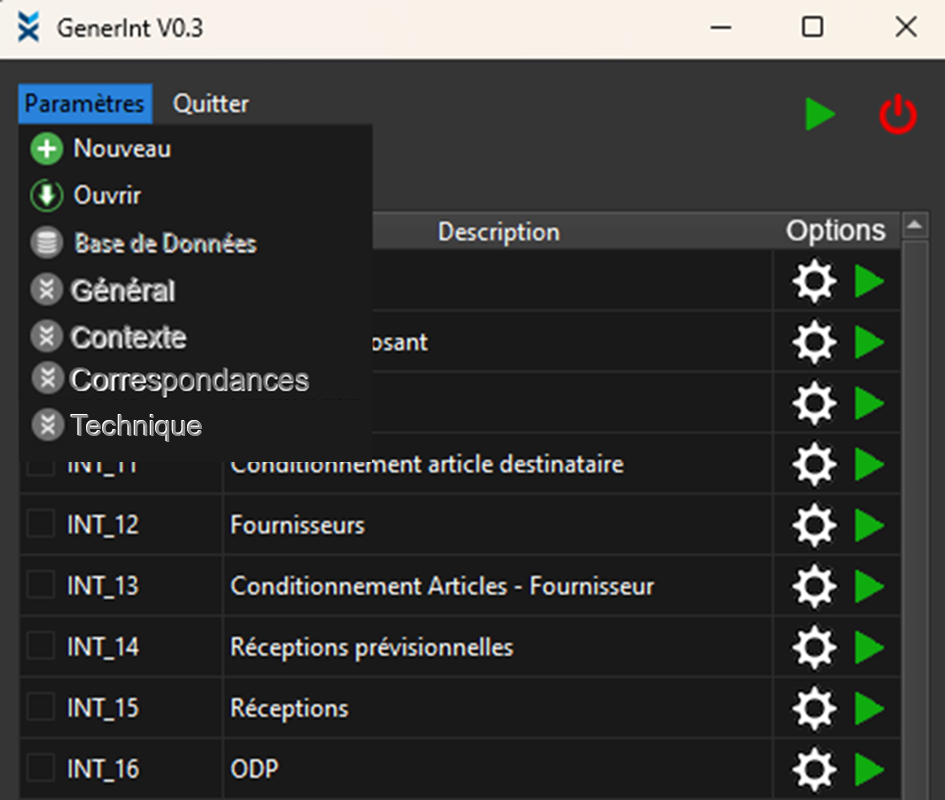


# Evolution

Les outils Odin et Exano sont déjà des applications graphiques, pour des questions d’homogénéité l’application devra utiliser la même charte graphique et la même ergonomie.

Le but sera de regrouper les paramétrages dans une base de données SQLite comme Odin.

## Ecran général



**7**

**9**

**8**

**6**

?

**5**

**4**

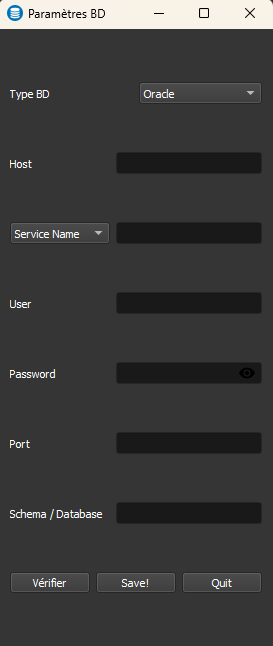
**3**

**2**

De la même manière que Odin, le menu « Nouveau » permet de créer un nouveau fichier de configuration (choix du nom et de l’emplacement) et « Ouvrir » permet de recharger une configuration existante.

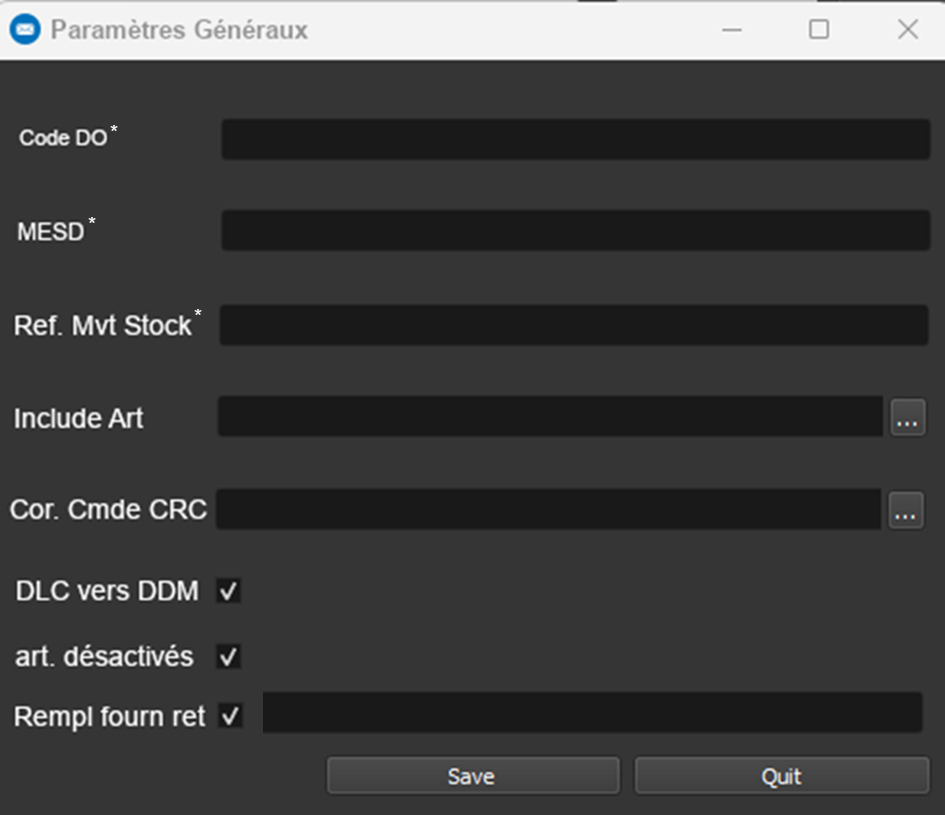
Le menu aide renverra sur la documentation de l’outil (choix du support à voir)

## Ecran de paramétrage BD



L’écran aura les mêmes fonctionnalités que celui d’Odin ou d’Exano avec cependant quelques modifications.  
Dans type BD ajouter AS400 en plus de MSSQL et Oracle. Du coup comme les 2 autres gérer les autres champs dépendants (se baser sur le connecteur existant).

## Ecran de paramétrage Général



On retrouve ici la partie paramétrage général avec :

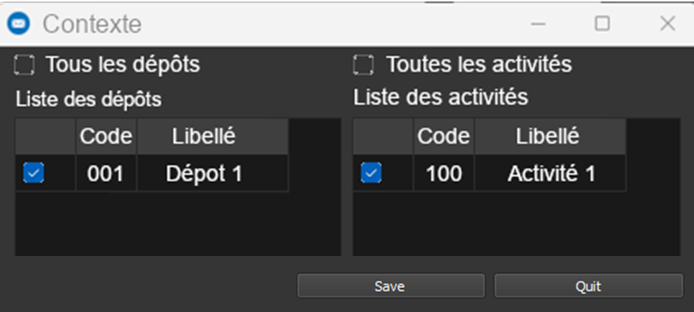
* Code donneur d’ordre (Pour interface 17)
* Motif entrée sortie diverse (Pour interface 17)
* Référence mouvement de stock (Pour Interface 17)
* Include Art (fichier csv contenant la liste d’articles en cas de reprise partielle par code article) (optionnel)
* Fichier de correspondance codes regroupement chargement (Pour interface 30. Optionnel)
* Bascule de la DLC vers la DDM (option)
* Reprise des articles désactivés (option)
* Remplacement du code fournisseur par celui indiqué en cas de GEI généré lors d’un retour (optionnel)

Il faudra mettre des infobulles sur chaque option avec ce qui est décrit précédemment

Le bouton « Save » enregistre et ferme la fenêtre

Le bouton « Quit » annule et ferme la fenêtre

## Ecran de sélection du contexte de reprise

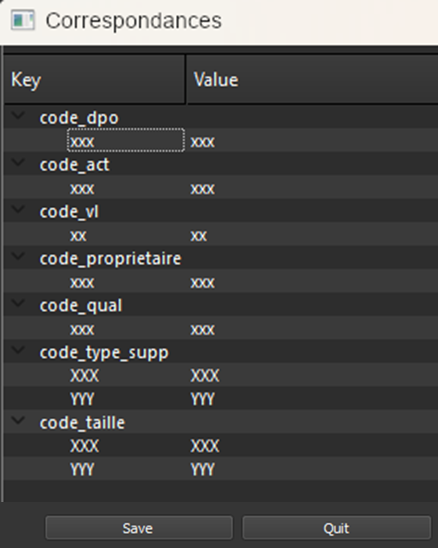


Le but est d’aller chercher les dépôts / activités actifs sur la BD pour les afficher (On peut se baser sur l’écran de stock\_a\_zero)

Le bouton « Save » enregistre et ferme la fenêtre

Le bouton « Quit » annule et ferme la fenêtre

## Ecran de paramétrage des correspondances



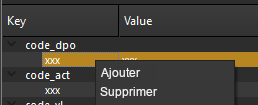
Ecran qui permet de paramétrer des correspondances sur des informations données (la clé sera remplacée par la valeur)

*Note : l’existant était géré via un fichier JSON, qui est facilement transposable en dictionnaire python. On peut peut-être stocker ça en BD SQLite sous forme JSON directement.*

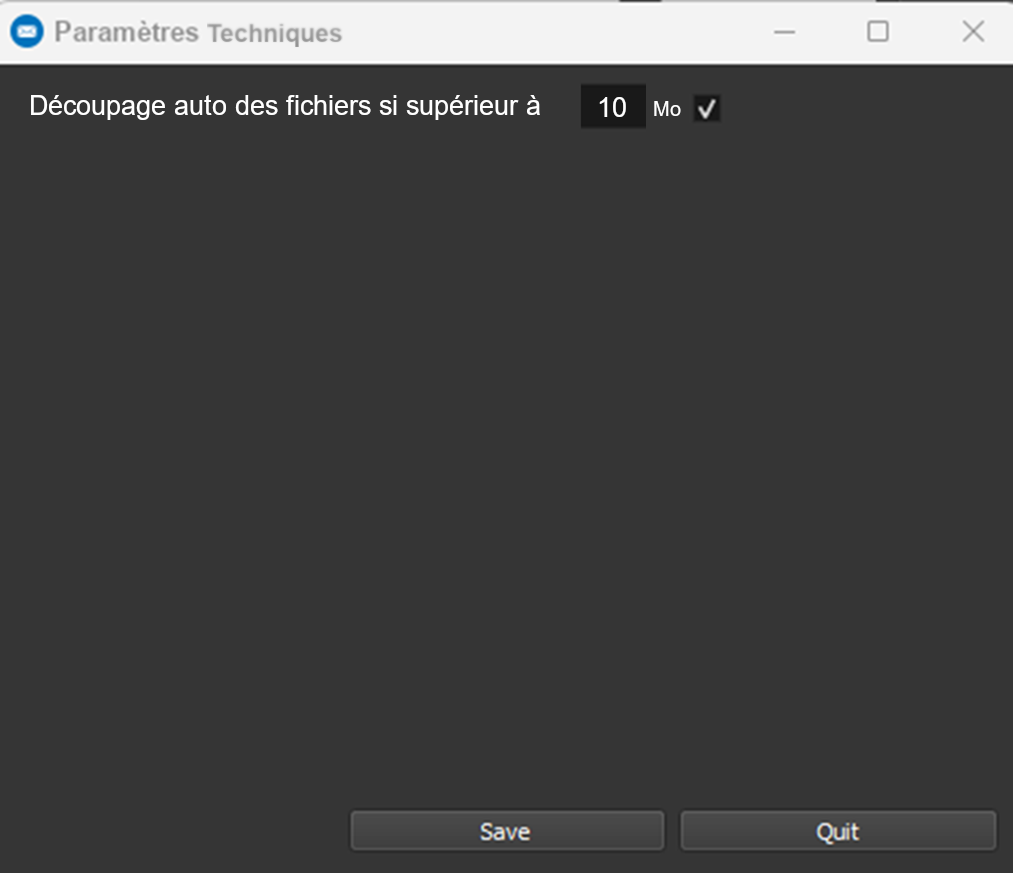
Le bouton « Save » enregistre et ferme la fenêtre

Le bouton « Quit » annule et ferme la fenêtre

On pourra via un menu contextuel ajouter ou remplacer des couples (clé / valeur)

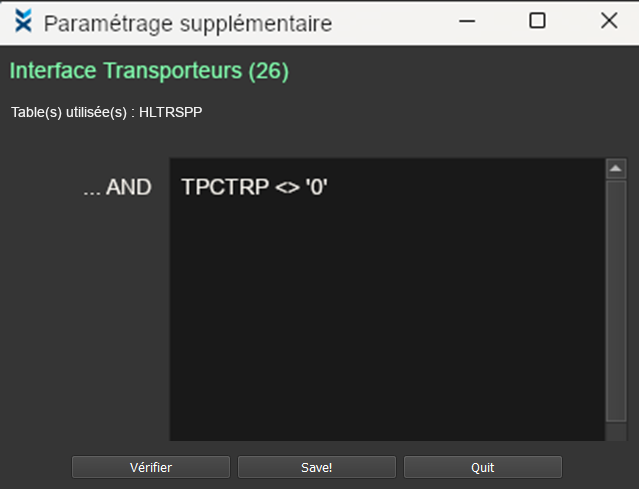


## Ecran de paramétrage technique



Ici, se trouve l’option de découpage des fichiers de sortie automatiquement si le fichier dépasse la taille paramétrée. On gardera le même de principe de découpage par rubrique quand c’est possible et sinon avant une rubrique 110 (cas 17 ou 47).  
Action activée par défaut.

## Ecran de paramétrage de chaque interface

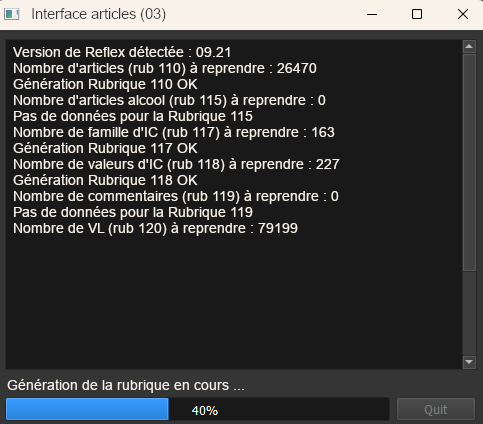


Cet écran permet de rajouter des paramètres supplémentaires sous forme d’instructions SQL si besoin. (La zone foncée ici est éditable).  
Il faudra afficher la ou les tables utilisées dans toutes les requêtes du programme d’interface pour aider l’utilisateur à choisir les champs à utiliser dans la queue de requête.  
Le bouton « Vérifier » permettra de tester la queue de requête (utiliser une requête simple derrière). Penser à gérer le cas si le paramétrage BD n’est pas fait (erreur dédiée).

Le bouton « Save » enregistre et ferme la fenêtre

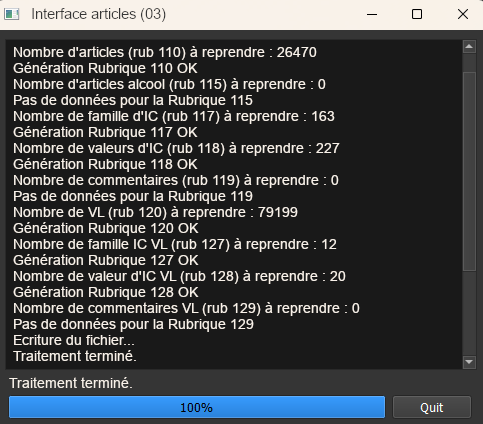
Le bouton « Quit » annule et ferme la fenêtre

## Lancement du programme de génération du fichier d’interface



On reprendra ici toutes les informations qui étaient affichées en mode console.  
La barre de progression se réinitialisera à chaque rubrique.

Le bouton quitter sera grisé pendant le traitement et redeviendra actif à la fin.



# Améliorations

## Interface 17

Afficher dans les logs le nombre de GEI, le nombre de supports et générer dans un excel ou csv le détail de la table HLSTOCP pour un type de stock 200 (code dépôt, code activité, code article, code VL, code propriétaire, code qualité, Quantité du stock en VL de base)

## Requêtes utilisées

Dans un premier temps afficher dans les logs, les requêtes utilisées pour la génération des interfaces.

Un deuxième temps serait de stocker tout ou partie des requêtes dans la base de données de paamétrage.

## Pouvoir appeler un programme de génération d’interface spécifique

Comme cas Dior ou Raja  
Il faudrait pouvoir ajouter une interface spé sans avoir à recompiler tout.

# Annexes techniques

## Extrait de source pour la sélection des dépôts activités

1. 1. def affichage\_depot(self):

2. 2.

3. 3. self.cursor.execute("SELECT DPCDPO, DPLDPO FROM HLDEPPP WHERE DPTOPD = '0' ")

4. 4. depot = self.cursor.fetchall()

5. 5.

6. 6. # Définition du tableau Qt5 pour afficher les résultats

7. 7. self.TabDepot.setRowCount(len(depot))

8. 8. self.TabDepot.setColumnCount(len(depot[0]) + 1)

9. 9. self.TabDepot.setHorizontalHeaderLabels([" ", "code", "libellé"])

10. 10.

11. 11. # Remplissage du nouveau tableau avec les résultats de la requête

12. 12. for i, row in enumerate(depot):

13. 13. for j, item in enumerate(row):

14. 14. self.TabDepot.setItem(i, j + 1, QTableWidgetItem(str(item)))

15. 15.

16. 16. # Ajout case à cocher dans chaque ligne

17. 17. self.checkboxdepot\_list = [] # On créer une liste des checkbox pour la checkall

18. 18. for row, dep in enumerate(depot):

19. 19. checkboxdepot = QCheckBox()

20. 20. checkboxdepot.setChecked(False)

21. 21. checkboxdepot.setStyleSheet("QCheckBox{padding: 5px;}")

22. 22. self.TabDepot.setCellWidget(row, 0, checkboxdepot)

23. 23. checkboxdepot.stateChanged.connect(self.affichage\_activite)

24. 24. self.checkboxdepot\_list.append(checkboxdepot) # ajouter la nouvelle instance à la liste pour la checkall

25. 25.

26. 26. # Affichage du tableau dans la fenêtre

27. 27. self.TabDepot.show()

28. 28. self.TabDepot.resizeColumnsToContents()

29. 29. self.TabDepot.resizeRowsToContents()

30. 30.

31. 31. # ========================================================================

32. 32. def affichage\_activite(self, state):

33. 33. if state == Qt.Unchecked:

34. 34. self.check\_all\_depot.setChecked(False) # on décoche checkAlldepot

35. 35.

36. 36. # On vide la liste des activités

37. 37. for row in range(self.TabActivite.rowCount()):

38. 38. self.TabActivite.removeRow(row)

39. 39.

40. 40. # Liste des dépots selectionnés

41. 41. self.depots = []

42. 42. try:

43. 43. for row in range(self.TabDepot.rowCount()):

44. 44. checkbox\_item = self.TabDepot.cellWidget(row, 0) # récupérer le QCheckBox de la première colonne

45. 45. if isinstance(checkbox\_item, QCheckBox) and checkbox\_item.isChecked():

46. 46. depot\_item = self.TabDepot.item(row, 1) # récupérer l'élément de la colonne "depot"

47. 47. if depot\_item is not None:

48. 48. depot\_value = depot\_item.text() # récupérer la valeur du champ "depot"

49. 49. self.depots.append(depot\_value)

50. 50.

51. 51. # On selectionne les activités correspondantes aux dépots sans doublon

52. 52. if str(self.type\_bd).lower() == 'mssql':

53. 53. query = "SELECT DISTINCT (DACACT) , ACLACT FROM HLACTIP JOIN HLDPPAP ON HLDPPAP.DACACT = HLACTIP.ACCACT " \

54. 54. "WHERE ACTOPD = '0' AND DACDPO IN ({0})".format(', '.join('?' \* len(self.depots)))

55. 55. else:

56. 56. lst\_bind = []

57. 57. i = 1

58. 58. for dp in self.depots:

59. 59. lst\_bind.append(":" + str(i))

60. 60. i += 1

61. 61. query = "SELECT DISTINCT (DACACT) , ACLACT FROM HLACTIP JOIN HLDPPAP ON HLDPPAP.DACACT = HLACTIP.ACCACT " \

62. 62. "WHERE ACTOPD = '0' AND DACDPO IN (" + ', '.join(lst\_bind) + ")"

63. 63. self.cursor.execute(query, self.depots)

64. 64. activite = self.cursor.fetchall()

65. 65.

66. 66.

67. 67. except:

68. 68. return

69. 69.

70. 70. # Définition du tableau Activité pour afficher les résultats

71. 71. self.TabActivite.setRowCount(len(activite))

72. 72. self.TabActivite.setColumnCount(len(activite[0]) + 1)

73. 73. self.TabActivite.setHorizontalHeaderLabels(["", "code", "libellé"])

74. 74. self.nb\_gei.setText(str(0))

75. 75.

76. 76. # Remplissage du tableau Activité avec les résultats de la requête

77. 77. for i, row in enumerate(activite):

78. 78. for j, item in enumerate(row):

79. 79. self.TabActivite.setItem(i, j + 1, QtWidgets.QTableWidgetItem(str(item)))

80. 80.

81. 81. # Ajout case à cocher dans chaque ligne

82. 82. self.checkboxact\_list = [] # On va créer une liste des checkbox pour la checkall

83. 83. for row, dep in enumerate(activite):

84. 84. checkbox\_act = QCheckBox()

85. 85. checkbox\_act.setStyleSheet("QCheckBox{padding: 5px;}")

86. 86. checkbox\_act.setChecked(False)

87. 87. self.TabActivite.setCellWidget(row, 0, checkbox\_act)

88. 88. checkbox\_act.stateChanged.connect(self.calcul\_nb\_gei)

89. 89. self.checkboxact\_list.append(checkbox\_act) # ajouter la nouvelle instance à la liste pour la checkall

90. 90.

91. 91. # Affichage du tableau Activité dans la fenêtre

92. 92. self.TabActivite.show()

93. 93. self.TabActivite.resizeColumnsToContents()

94. 94.

95.