Acesso a API dos Componente de Negócios:

A API dos componentes de negócio(*ComponentAccessor*) permite consultar, atualizar e deletar registros de uma determinada tabela. Para acessar a API, utiliza-se o método *access()* da classe *SystemAutomator* passando como parâmetro o identificador “component”, conforme código a seguir:

```
ac = SA.access("component");
```

Víncular API a um componente de negócio:

Para vincular a API a um determinado componente de negócio utiliza-se o método *attach()* passando como parâmetro o nome do formulário do componente de negócio.

```
ac.attach("cliente");
```

Realizar consultas no componente de negócio:

Para realizar consulta no componente de negócio e retornar registros, utiliza-se o método *getRecords()*, conforme exemplo a seguir:

```
ac.getRecords();
```

O retorno da consulta corresponde a uma lista de itens da classe *TableDataSet. Um objeto da classe TableDataSet corresponde a um registro do componente de negócio. Para acessar os valores de seus campos* deve-se utilizar o método *getColumnValue()* passando como parâmetro o nome do campo. O código a seguir mostra um exemplo de como iterar os registros retornados na consulta e como exibir valores de seus campos.

```
for ( item : ac.getRecords() )
{
        print( "id: " +  item.getColumnValue( "id" ) );
        print( "nome: " +  item.getColumnValue( "nome" ) );
}
```

Para iterar entre uma lista de valores de um campo específico utiliza-se o método *getValues()* passando como parâmetro o nome do campo.

```
for ( valor : ac.getValues(“nome”) )
{
        print( "nome: " +  valor );
}
```

Adicionar filtro a consulta:

Para criar um filtro utiliza-se o método *filter()* e para adicionar condições ao filtro criado utiliza-se o método *addCondition()* que pode ser usado passando três parâmetros: nome da coluna, operador e valor, conforme demonstrado no exemplo 01. O método *addCondition()* também permite passar uma cláusula inteira como parâmetro, conforme pode ser visto no exemplo 02.

Exemplo 01:

```
ac.filter().addCondition("idade", ">" , "10");
```

Exemplo 02:

```
ac.filter().addCondition("idade > 10");
```

Para Criar mais de uma condição no filtro basta fazer chamadas consecutivas do método *addCondition()*, vide exemplo 03:

```
ac.filter().addCondition("idade", ">" , "10")
          .addCondition(“nome”,”=”,”Pedro”);
```

Por padrão é utilizado o operador lógico “AND” entre as condições. Caso seja necessário utilizar o operador lógico “OR”, a API permite criar um grupo de condições utilizando o operador desejado. Para isso utiliza-se o método *addGroup()* que recebe como parâmetro um objeto da classe *GroupFilter*. Um objeto da classe *GroupFilter* pode ser criado de uma forma simples através dos métodos *groupOR()* e *groupAND()*, onde cada um deles já tem um operador lógico definido. Veja exemplo a seguir:

```
ac.filter().addGroup( ac.groupOR().addCondition("idade", ">" , "10")
                                  .addCondition("nome","=","Carlos") ) ;
```

A partir dos métodos *groupOR()* e *groupAnd()* é possível montar filtros com diversos grupos e operadores diferentes. A tabela a seguir monstra alguns exemplos de filtros com grupos:

Filtro	Código
(idade > 10 and nome = ‘Carlos’)	*ac.filter().addGroup(ac.groupAND()* *.addCondition("idade", ">" , "10")* *.addCondition("nome","=","Carlos")) ;*
((idade < 10 or name = ‘Alberto’) && (idade > 10 or name = ‘João’))	*ac.filter().addGroup(ac.groupAND()* *.addGroup(ac.groupOR()* *.addCondition("idade < 10")* *.addCondition("nome", "=", "Alberto"))* *.addGroup(ac.groupOR()* *.addCondition("idade > 10")* *.addCondition("nome", "=", "João")));*

Inserir registro no componente de negócios:

Para adicionar um registro no componente de negócios é necessários setar valores para os seus campos, para isso utiliza-se o método *fields()* seguido do método *addValues()*. O método *addValues()* permite passar como parâmetro uma sequência de duplas que correspondem ao nome e ao valor do campo, conforme exibido no Exemplo 1:

```
ac.fields().addValues( new String[]{ "nome", "Jorge", "idade", "10" } );
```

Também é possível adicionar campos e valores de forma independente conforme Exemplo 2:

```
ac.fields().addValue( “nome”, “Jorge”).addValue(“idade”,”10”);
```

Caso o campo *Primary Key* do componente de negócio seja do tipo inteiro, não há a necessidade de adicionar valor a esse campo, pois o mesmo receberá o próximo número da sequência (MAX + 1).

Após definir os campos e valores para o novo registro é possível inseri-lo na tabela do componente de negócio a partir do método *addRecord()*, vide exemplo a seguir:

```
ac.addRecord();
```

Atualizar registros de um componente de negócios:

Para atualizar os dados em uma tabela é necessário setar os valores para os campos que devem ser atualizados e também setar os filtros caso seja necessário.

```
ac.fields().addValues( new String[]{ "nome", "Júlio", "idade", "20" } );

ac.filter().addCondition( "id = 1" );
```

Após definição dos campos e filtros utiliza-se o método *updateRecords() para atualizar os registros do componente de negócio*, conforme exemplo a seguir:

```
ac.updateRecords();
```

Remover registros de um componente de negócios:

Para remover registros de um componente de negócios basta definir um filtro e chamar o método *deleteRecords(),* vide exemplo a seguir:

```
ac.filter().addCondition( "id = 1" );

ac.deleteRecords();
```

Limpar o filtro:

Para limpar o filtro, utiliza-se o seguinte código:

```
ac.filter().clear();
```

Limpar os campos:

Para limpar os campos, utiliza-se o seguinte código:

```
ac.fields().clear();
```