Optimizing the Performance of a Mobile Handheld Application in SAP NetWeaver Mobile

Applies to:

SAP NetWeaver Mobile 7.10. For more information, visit the Mobile homepage

Summary

This document talks about some best practices for optimizing on performance, which should be taken into account while developing a SAP NetWeaver 7.10 Mobile Application for Handhelds.

Author: Nipun Dev

Company: SAP Labs India

Created on: 30 November 2009

Author Bio

Nipun Dev is working with SAP NetWeaver Mobile organization in SAP Labs India.

Table of Contents

Optimizing the Performance of a Mobile Handheld Application

To ensure that a Mobile application for Handhelds has optimal performance, there are some best practices that are followed.

1. Service Component

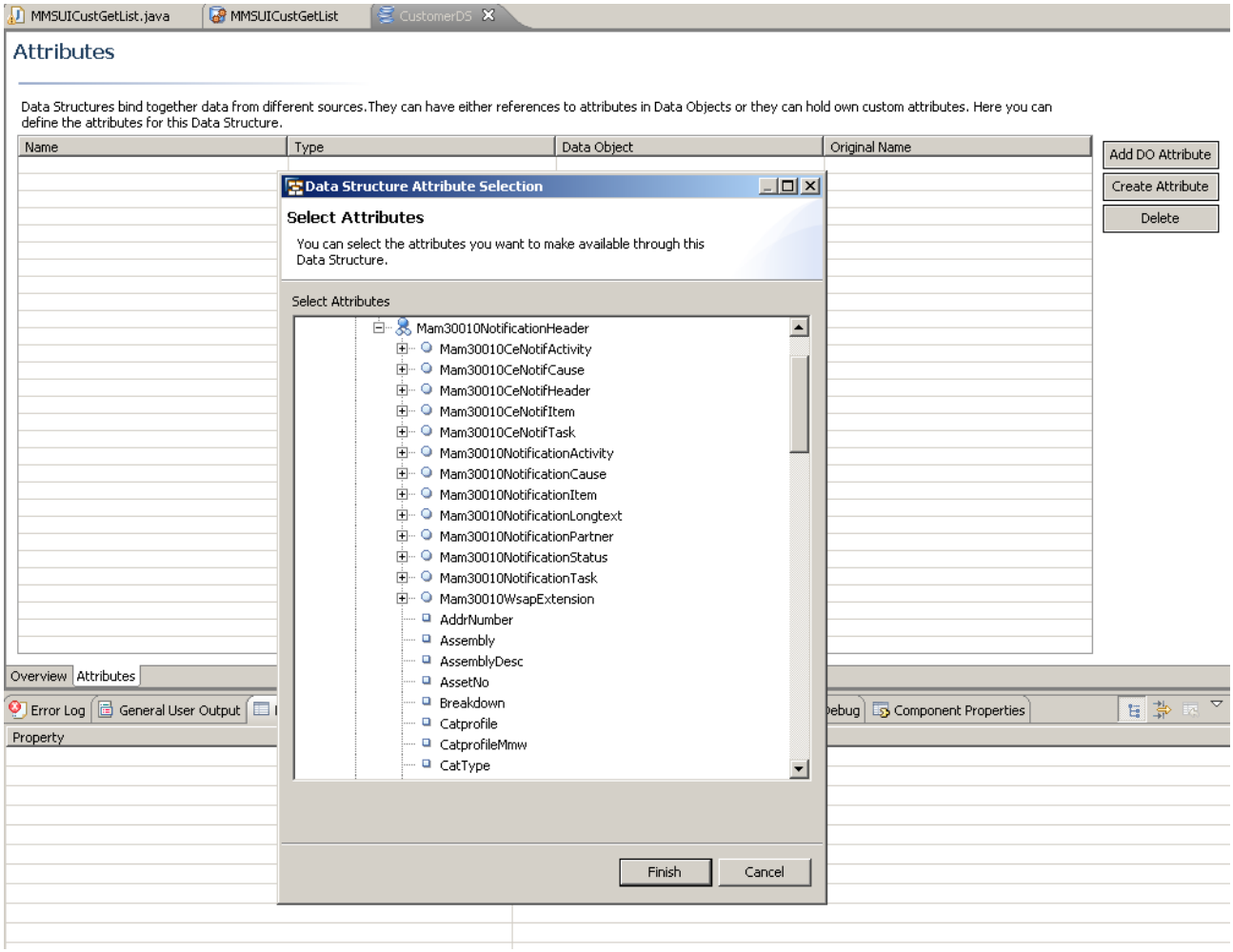
This section covers do's and don'ts for query and service operations:

- Avoid creating query initializations. Initialize queries and other stuff lazily. E.g. Do not store the queries as field in the component controller *(e.g. private GenerateEmployeeOverviewQuery overviewQuery = ...OcaRoot.getInstance ()...)* since this loads the complete query metadata when the application is started (or more specific when the class is loaded during application start). Try to guard the access to the query objects through methods instead, so they will be initialized when they are needed.
- Avoid using *do...while like iterators*, iterations on a query result using count method on a collection. *Collection.Count* on a Query result will lead to performance degradation. If the requirement is to check the collection has any element, do the following.

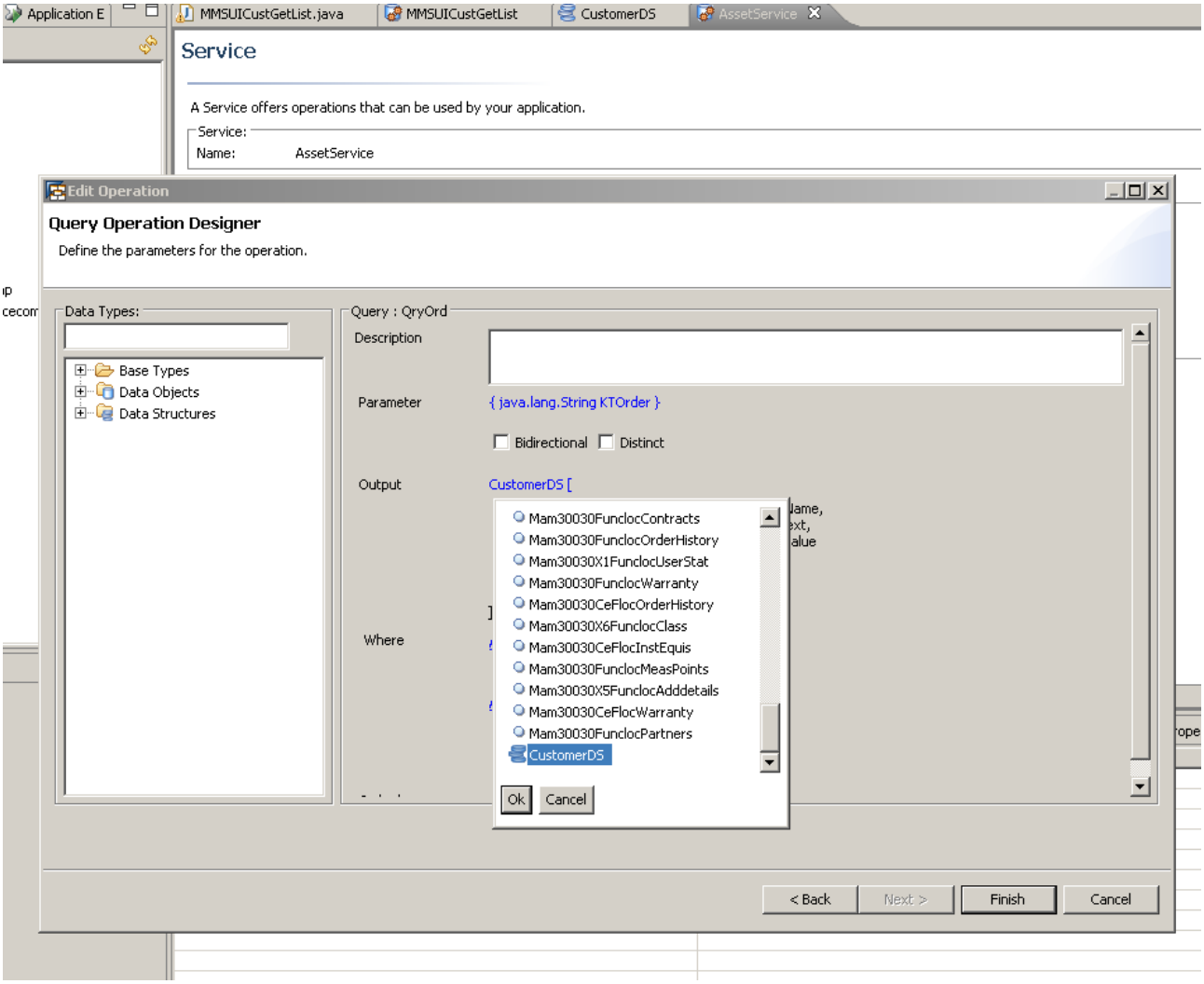
```
Collection col = Query.execute ();
Iterator it = col.iterator ();
If (it.hasNext ())
{
    //Collection is not empty. Do the application operations
}
```

To limit the volume of data required to be fetched from the database, make use of data structures as query results. You can model a data structure with the necessary attributes from a data object node and use the same in the query result. This will reduce the volume of Data fetched from the Database.
Using a DataObjectNode as a Query result is as expensive as doing a <<Select * from a table>>.

Creating Data Structures from necessary attributes (Figure 1):

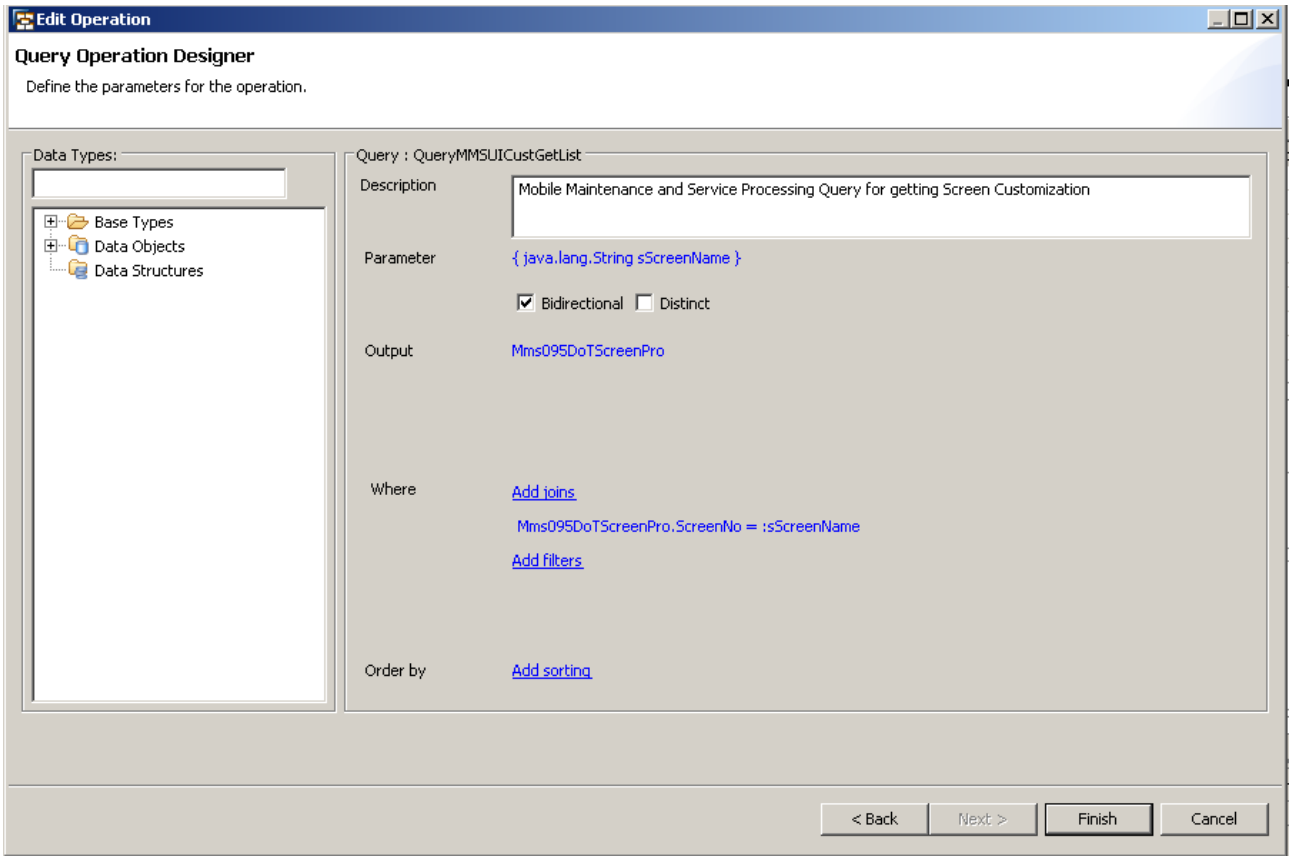


Using Data Structures as Query Output (Figure 2)



- If you want to traverse through a large query result, it is recommended that you choose the Bidirectional option while creating a query to ensure that the iteration takes place faster. However if the query result contains only few fields, it is always best to traverse linearly.

(Figure 3)



- Completely avoid OR conditions! This always results in temporary table creation and long execution time.
- It is always recommended to create all structures for the current business and then use *OCARoot.commit ()*. Otherwise this results in multiple database operations on the same object.

  Use *OCARoot.commit ()* for saving the data changes with care, as it results in a db commit which is always expensive. First create all structures for the current business case in memory if possible (root and child nodes). Otherwise it will result in multiple database operations (first an insert and second an update to change the internal state). Keep in mind that querying data will flush all in-memory objects to database. So avoid executing queries while creating your in-memory objects.
- Define appropriate indexes for the data object node during design time. *SYNCKEY* and *PSYNCKEY* are the default indexes.

  Most of the application search will be happening on an attribute which is used by the end user or on backend key fields.
  For E.g.: Searching a Business partner will be based on a Partner Name or Partner ID. In this case it is advisable to set the Partner Name or Partner ID as Indices.

When to consider index creation

- *SELECTs* with an *ORDER BY* clause to avoid temporary table creation if the column has high selectivity
- Multi-table *SELECTs* with *JOIN* and/or projection conditions
- Conditions connected with *AND*-operator
- Filter on columns with a high percentage of distinct values (high selectivity) compared to the overall number of rows in the table

However, index creation has the following drawback:

- Insert, Update and Delete performance will degrade with every index.

More information:

- *Performance Tuning*

2. UI Component

- Import only the relevant model classes from the service component to the UI Component.
- If the model class is required only for a specific view, bind the model class to the specific view context. Do not bind the class at the parent component level and then map the view context to it.
- If you are reusing the model classes across views, create and bind the model classes at component context level only. Then map the view context nodes to the component context. This ensures especially for a read that the data is read once and used by many.
- Keep only the required context node or attributes at the view level.
- In productive systems, it is recommended that severity level of trace messages is set to *ERROR.*
- Avoid using the synchronization log to store all the inbound and outbound messages. This should ideally be used for internal debugging only.
- Use typed context node access if there are not many context nodes. Check the Typed Context Node property as *false* if there are too many Context Nodes.

  Typed context generation helps you access the context nodes & attributes with named methods for easy use. E.g. the context node can be accessed as *wdContext.node<<DONodeName>>* & attribute as *wdContext.node<<DONodeName>>.get<attributename>.*

  But in case of too many context nodes this becomes expensive due to performance reasons then its better to use generic methods like:-

  wdContext.getNode(<<DONodeName>>).getAttribute(<<attributeName>>)
- Use APIs to find an element in the context node collection. Typed generated APIs can be used to refer to currently selected context node element.
- Avoid holding the query results in context node or member variable in the component controller if it is no more required.
- Keep the number of UI elements in a view to a minimum. Try to break the use case in multiple views to enable lighter views for faster rendering.
- Avoid using multiple tabs or transparent containers inside a single view. You can break the tabs into different views and use minimal number of transparent containers.
- You can create a separate query for sorting elements to improve the response time.

General Practices

- Use Db2e for better performance. Refer to SAP Note Number: 1121247
- Keep only relevant data on the device to keep the application database size optimal.
- It is recommended to test the application on a handheld device in addition to testing it on the Windows desktop simulator provided with NetWeaver Developer Studio (NWDS).This is done to assess the actual performance using a handheld device.
- It is recommended to use 128MB RAM and 128 MB fast Flash ROM devices. The performance depends on the speed of the device memory.

Related Content

Developer's Guide - SAP NetWeaver Mobile 7.10

Tutorial - Developing Mobile Handheld Applications

For more information, visit the Mobile homepage

Copyright