Client Extension - the Swiss Knife for Oracle Projects

Abhishek Chandan
Ideametrics, LLC

Executive Summary

Client extensions provide the ability to extend base functionality provided in projects module. This paper describes how various companies use client extensions to implement complex business logic without having to customize delivered functionality.

This white paper presents the following case studies. Each case study is described with high level requirements, design, code snippets of actual implementation logic, and lessons learnt.

Asset Capitalization – This case study covers two different client extensions – asset creation and asset assignment client extension. One of the companies utilizing these extensions capitalizes approximately three billion dollars of assets using these extensions.

Transaction Control –This case study illustrates how custom validation logic is applied to control types of expenditures on a project.

Project Verification – This case study illustrate how custom validation logic is applied to control whether or not project status can be changed to “Approved.”

Introduction

Although, Oracle Projects provides a lot of flexibility in processing projects, many companies have business requirements that are unique to their company. To address these unique requirements, Oracle Projects provides several client extensions that enable businesses to extend the functionality of the product to implement and automate company-specific business rules.

Client extensions allow automation of business rules within the standard processing flow of Oracle Projects, without having to customize the software. These rules are designed and implemented using PL/SQL procedures; these procedures are called during specific points in the standard processing flow of Oracle Projects. Each client extension has an associated seeded package and package body that must be modified to implement these business logics.

Implementing Client Extensions

The implementation of client extensions primarily consists of the following three steps:

- Determining business requirements
- Designing client extension logic
- Writing PL/SQL procedures

Each of these steps requires a specific expertise. The analysis and design portions require an implementation team member who is functionally knowledgeable about the company specific business rules, the implementation of Oracle Projects within the company, and the conceptual flow of the client extensions. The PL/SQL coding portion requires an implementation team member who is technically knowledgeable with PL/SQL and the Oracle Projects data structures.

Determining Business Requirements

The first step of implementing client extensions is to determine if a client extensions a truly needed. Typically, this process involves the following steps. These steps are part of most standard project implementation methodologies.

Step 1: Clearly define and document your company specific business requirements and rules.

Step 2: Determine if these business rules are handled by the standard functionality of Oracle Projects.

Step 3: For those business rules not handled by the standard functionality, review the client extensions and determine if the client extension can help address the specific business rules, based on your documented business requirements.

Example of Determining Need for Client Extension

Let's look at an example of this process.

Step 1: Company has defined this policy:

You can only charge supplies to overhead projects

Step 2. You review the standard functionality of transaction controls in Oracle Projects and find that you can implement what can be charged or not charged to a specific project or task using transaction controls. The rule regarding supplies is applicable to all projects that are not overhead projects. You could implement this rule by procedurally defining transaction controls for every non-overhead project, but that is impractical.

Step 3. You decide that you can use transaction control extensions to implement this company-wide policy regarding supplies.

Designing Client Extensions

Designing client extension is the most significant part of implementing client extensions. A careful, thorough design and specifications in this stage will result in manageable and simple coding phase and a more successful client extension implementation. This design cycle involves the following aspects:

Step 1: Understand the appropriate client extension, including its intended purpose, its processing flow, the predefined place that it is called, and its input values. Each essay in this chapter describes the processing flow for a client extension. Since each client extension may be processed differently by Oracle Projects, you should carefully review the flow and additional design information we provide for each extension.

Step 2: Define and document the requirements and logic of your business rules under all possible conditions. This logic includes the required inputs, the calculations performed, and the corresponding outputs.

Step 3: Determine the data elements required to drive your rules and how you will select or derive each of the required elements. Define additional implementation data and document additional business procedures of using the system based on the requirements of your business rules.

Step 4: Step through various business scenarios to ensure that your logic handles each condition as you expect. You can use these scenarios as test cases when you test your actual client extension definition and procedure.

Step 5: Hand off detailed specification of logic to the technical resource who will write the PL/SQL procedure.

Determining Data Elements

Predefined Parameters
Oracle Projects provides predefined parameters for use in client extensions. The program which calls and executes the client extension passes in values for the predefined parameters, which define the context of what transaction is being processed.

Derived Parameters
Additional parameters may be derived from the predefined parameters. For example, a client extension may have a predefined parameter of PROJECT_ID, which is the identifier of the project. Your business rule needs project type, so you derive the project type from the PROJECT_ID.

Descriptive flexfield (DFF) segments may be used to hold additional data as inputs into your rules. The PL/SQL procedure may select from the descriptive flexfield segment column which holds the appropriate input value.

Example of Designing Client Extension
Let's use our earlier transaction control extension example to illustrate these design steps.

Step 1: After studying transaction control extensions, you have decided to use transaction control extensions to implement the following policy:

You can only charge supplies to overhead projects

Step 2: You define the logic for the transaction control extension as:

```
IF      charging supplies
THEN    IF      charging to overhead projects
        THEN    OK
        ELSE    error message
You can only charge supplies to overhead projects
ELSE    OK
```

Step 3: You now determine the required data elements to identify what transactions are supplies and what projects are overhead projects.

You decide that the expenditure type of Supplies specifies the type of charge, and that the project type of Overhead specifies the type of project.

The transaction control extension predefined parameters include expenditure type and project ID. You can derive the project type of the project using the project ID.

You have already defined these values as implementation data.

Project Type: Overhead

Expenditure Type: Supplies

When using these data elements, the logic for the transaction control extension is as follows:

```
IF Expenditure Type = Supplies
THEN    IF Project Type = Overhead
```

THEN OK
ELSE error message
You can only charge supplies to overhead projects
ELSE OK

Step 4: You step through several scenarios of different types of charges to different types of projects. Your logic handles all of the scenarios.

Step 5: You are ready to hand off this specification to your technical resource.

Using Template Procedures
Oracle Projects provides you with template procedures for each client extension. Each procedure contains predefined parameters that are passed into the procedure by the program that calls the procedure; these predefined input parameters cannot be changed.

The “Oracle® Projects APIs, Client Extensions, and Open Interfaces” user manual lists each client extension and its predefined template procedure filenames. The template procedure files are stored in the Oracle Projects admin/sql directory.

Suggestion: Review the appropriate files before you design and implement a client extension. They provide a lot of useful information, including the predefined input parameter list and example case studies.

Suggestion: You should make a copy of these template files in a directory used by your company to store code that you have written. You should make changes to these copies of the files instead of writing directly into these template files. These template files will be replaced when the software is upgraded between releases. Use your modified files to reinstall your procedures after an upgrade to a new release of Oracle Projects.

Writing Logic in Your PL/SQL Procedures
The logic in the PL/SQL procedures is based on the functional specifications created during the design process. Before you begin to write the client extension PL/SQL procedures, you should have a clear understanding of the client extension procedures; including the inputs and outputs, the error handling of the extension, along with any example procedures provided for each extension. Read the appropriate client extension essays and template procedures to obtain detailed information about the client extensions.

As you determine how to best write the client extension, you should consider these issues:

- Can I derive every derived input parameter based on the data structures known?
- What outputs should the client extension return?
- How does the client extension handle exceptions?
- Are there procedures which I can write which are reusable across similar client extensions?
- How I can write logical, well commented code that is easy to maintain and debug?
- How do I test and debug this client extension?
- Are there any performance considerations in the client extension? If so, what are they and how do I address them?

Attention: You must not commit data within your PL/SQL procedure. Oracle Projects processes that call your procedures handle the commit logic.

Storing Your Procedures
After you write your procedures and ensure that the specification file correctly includes any procedures that you have defined, you need to compile and store the procedures in the database in the Applications Oracle username. You must install the package specification before the package body.

The syntax for compiling and storing PL/SQL procedures is included in the template procedure files. Assuming you have written your procedures using copies of these template procedure files, you can use these steps to compile and store your procedures:

Change to the directory in which your files are stored (use the command that is appropriate to your operating system)

$ sqlplus <apps username>/<apps password>

SQL> @<spec_filename>.pls <apps username> <apps password>

SQL> @<body_filename>.pls <apps username> <apps password>

For example, you use the following commands to install your transaction control extensions (assuming your Oracle Applications Oracle username/password is apps/apps):

$ sqlplus apps/apps

SQL> @PAXTTXCS.pls apps apps

SQL> @PAXTTXCB.pls apps apps

If you encounter compilation errors in trying to create your packages and its procedures, you must debug the errors, correct your package definitions, and try to create your packages again. You must successfully compile and store your package and its procedures in the database before you can use the client extensions in Oracle Projects.

Testing Your Procedures
After you have created your client extension procedures, you must test your client extension definitions within the processing flow of Oracle Projects to verify the results are as you expect.

Case Study: Asset Capitalization

Business Scenario
Client has over fifteen thousand (15000) active projects. All accumulated charges for the projects must be capitalized on a monthly basis. The capitalization process requires that charges be correctly capitalized among 150 or so asset categories. To make things further complicated, it is possible that one project may generate asset lines for more than one category in any month.

Analysis
Standard functionality is sufficient in the sense that users may manually go thru' asset creation and asset assignment process manually in accordance with business requirements. However, the volume of data as well as granularity of asset classification makes the task too onerous and risky. Since the capitalization rules were very rigid with little to no manual decision making involved, project team decided to use client extension to automate the process.

The actual logic was implemented using a combination of two client extensions – Asset Lines Processing Extension and Asset Assignment Extension. These extensions are called by "PRC: Generate Asset Lines" concurrent program. As part of "PRC: Generate Asset Lines" execution, the asset lines processing extension logic creates new project assets on as needed basis, and the asset assignment extension assigns expenditure lines to project assets.

Code Snippet
Package: PA_CLIENT_EXTN_ASSET_CREATION

Business Logic:

(extension called for each project by Oracle)

```
For each combination of expenditure_type and period charged on the project

     Check if there is an existing project asset.

      If there is no existing project asset

            Call Pa_Project_Assets_Pub.add_project_asset api
            Store expenditure_type in attribute1
            Store period in attribute2
```

Code Snippet
Package: PA_CLIENT_EXTN_GEN_ASSET_LINES

Business Logic:

(extension called for each expenditure item by Oracle)

Select project_asset
where ATTRIBUTE1 = expenditure_type and ATTRIBUTE2 = CDL period

Assign expenditure_item to selected project

Case Study: Transaction Control

Business Scenario
Client uses WebADI project transaction entry method to enter various pre-approved manual expenditure to projects. Business rules require that all manual expenditure must populate attribute1 to facilitate certain reporting requirements.

Analysis
During initial analysis, it was assumed that this may be as simple as setting up a mandatory descriptive flexfield segment to force the users to populate that value. During testing, users realized that making a DFF segment mandatory result in all payables and inventory transactions to be rejected as well.

The alternative was to make the DFF definition context sensitive. A context sensitive DFF segment resolved the issue with payables and inventory transactions. It also worked well with online miscellaneous expenditure entry form. However, due to the technical limitation of WebADI architecture (a known bug euphemistically referred as enhancement request, see ER2809958), this was not feasible. At this point, the project team decided that transaction control client extension will be used.

Code Snippet
Package: PATCX

Business Logic:

(called for each transaction by Oracle)

```
IF     (X_calling_module = 'PAXTRTRX')
   AND (X_transaction_source = 'XXXX Project Spreadsheet')
   AND (X_attribute1 IS NULL) THEN
   X_msg_type := 'E';
   X_outcome := 'PATCX_ATTRIBUTE_NULL';
END IF;
```

Case Study: Project Verification

Business Scenario
Client has a business rule that requires that each approved project must have a base-lined budget.

Analysis
There was no seeded functionality that met this requirement. Project team evaluated two options – setting up project status change workflow or using project verification extension. Since, enabling the workflow required a change to client extension anyway, project team decided to use client extension functionality instead.

Code Snippet
Package: PA_CLIENT_EXTN_PROJ_STATUS

Business Logic:

(called whenever user attempts to change project status)

```
IF     (new_system_status_code = 'APPROVED') THEN
        Check if baselined budget exists.
              If yes, allow status change
              If no,raise error
```

Conclusion

Client Extensions can be used in variety of ways to meet business requirement without customizing delivered Oracle code. Oracle provides more than fifty (50) client extensions in release 11.5.10.2 to allow end users to augment and tune delivered functionality to meet business requirements.

Contact Information for Author
Email: achandan@ideametrics.com