Architecture & Design

Cloud Processing Framework

3.0.0

Prepared for:



Table of Contents

1 Introduction 6

1.1 Purpose 6

1.2 Scope 6

1.3 References 7

1.4 Overview of Document 7

2 System Overview 8

2.1 Project Perspective 8

2.2 System Context 9

2.3 General Constraints 9

2.4 Assumptions and Dependencies 9

3 System Architecture 10

3.1 Enterprise Viewpoint 10

3.1.1 Roles 10

3.1.2 Policies 10

3.2 Information Viewpoint 11

3.3 Computational Viewpoint 12

3.3.1 Component Interactions 19

3.3.1.1 Client Job Submission 19

3.3.1.2 Batch Job State Transitions 20

3.3.1.3 Executor Service 21

3.3.1.4 Batch Job Pre-Processor 22

3.3.1.5 Batch Job Scheduler 23

3.3.1.6 Batch Job Post-Processor 24

3.3.1.7 Worker Batch Job Request Group Execution 25

3.3.2 Interface Specifications 27

3.3.2.1 Resource Representations 28

3.3.2.2 Client Web Services 34

3.3.2.3 Internal Web Services 42

3.4 Engineering Viewpoint 50

3.5 Technology Viewpoint 51

3.5.1 Component Technologies 51

3.5.2 Interface Protocols 51

3.5.3 Servers 52

3.5.4 Database Instances 52

3.5.5 Network Shares 52

4 Data Design 52

4.1 Data Security 52

4.2 Data Model Design 53

4.3 Warehouse Publication and Consumption 54

5 Human Interface Design 54

6 Technical Requirements 54

6.1 Performance Requirements 54

6.2 Quality Assurance 55

List of Tables

Table 1 – Scope – Software Components 6

Table 2 – References 7

Table 3 – Enterprise Viewpoint Roles 10

Table 4 – Enterprise Viewpoint Policies 11

Table 5 – CPF Class Descriptions 12

Table 6 – Media Types and file extensions 27

Table 7 – Component Technologies 51

Table 8 – Component Technologies 52

Table 9 – Servers 52

Table 10 – Database Instances 52

Table 11 – Database Roles 53

Table 12 – Database Permission Flags 53

Table 13 – CPF Data Security Matrix 53

List of Figures

Figure 1 – CPF Information Viewpoint 11

Figure 2 - Parallel Processing Diagram 13

Figure 3 – Job Group Decomposition 16

Figure 4 - Parallel Task Execution 18

Figure 5 – Client Job Submission Sequence Diagram 19

Figure 6 – Batch Job State Diagram 20

Figure 7 Batch Job Pre-Process Sequence Diagram 22

Figure 8 – Batch Job Scheduler Sequence Diagram 23

Figure 9 Batch Job Post-Process Sequence Diagram 24

Figure 10 – Worker Batch Job Request Group Execution 26

Figure 11 – Business Application Parameters 35

Figure 12 – Business Application Version Parameters 35

Figure 13 – Business Application Multiple Requests Form Parameters 36

Figure 14 – Business Application Multiple Requests Form Parameters 36

Figure 15 – Business Application Single Request Form Parameters 37

Figure 16 – Business Application Multiple Requests Form Parameters 38

Figure 17 – User Parameters 39

Figure 18 – User Applications Parameters 39

Figure 19 – User Application Parameters 40

Figure 20 – User Application Jobs Parameters 40

Figure 21 – User Jobs Parameters 40

Figure 22 – User Job Parameters 41

Figure 23 – User Job Attributes 41

Figure 24 – User Job Results Parameters 42

Figure 25 – User Job Result Parameters 42

Figure 26 – User Authenticated Attributes 42

Figure 27 – User Attributes Parameters 43

Figure 28 – WebADE User Attributes 43

Figure 29 – User is Member of Group Parameters 44

Figure 30 – GroupMembership Attributes 44

Figure 31 – User Can Perform Action Parameters 44

Figure 32 – ActionPermission Attributes 44

Figure 33 – User Can Access Resource Parameters 45

Figure 34 – ResourcePermission Attributes 45

Figure 35 – Worker Ping Parameters 46

Figure 36 – WorkerPing Attributes 46

Figure 37 – Next Batch Job Execution Group ID Parameters 46

Figure 38 – ExecutionGroupId Attributes 46

Figure 39 – Batch Job Execution Group Parameters 47

Figure 40 – ExecutionGroup Attributes 47

Figure 41 – Batch Job Execution Group Results Parameters 48

Figure 42 – ExecutionGroupResults Attributes 48

Figure 43 – ExecutionGroupResultsConfirmation Attributes 49

Figure 44 – Batch Job Request Opaque Output Data Parameters 49

Figure 45 – Batch Job Request Opaque Input Data Parameters 49

Figure 46 – OpaqueOutputDataResults Attributes 50

Figure 47 – Engineering Viewpoint 51

Figure 48 – Client Web Services Resource Tree 54

# Introduction

## Purpose

The purpose of this document is to describe the requirements, architecture and design of the Cloud Processing Framework (CPF) system.

The contents of this document are intended to be used by the following audiences.

* During application design, development, implementation and deployment.
  + Ministry staff responsible for approval of the architecture, design and deployment specifications.
  + Ministry staff responsible for deployment of the application to test and production.
  + Ministry (Business Area) staff responsible for testing the application.
  + Development staff responsible for implementation, testing and deployment of the application to the delivery server.
  + External application staff responsible for interfacing to the application.
* After implementation.
  + Ministry staff responsible for managing the application components.
  + Maintenance staff responsible for fixing defects and minor enhancements.
  + Development staff responsible for extending the application with new features and components.
  + External application staff responsible for interfacing to the application.

## Scope

The scope of this document is the architecture and design of the Cloud Processing Framework (CPF) system.

**NOTE:** This document does not cover the architecture and design of the business applications deployed within the CPF. Each of these business applications must have their own architecture and design documents.

The following table summarises the software components that are described in this design document.

Table 1 – Scope – Software Components

| Software Component | Description |
| --- | --- |
| CPF Database | The operational database environment containing the data required by the CPF application. |
| CPF Web Application | The web application containing the client and internal web services, administration application and scheduler component. |
| CPF Worker | The web application containing workers that execute the business application requests. |
| CPF Client API | The Java & JavaScript API used by clients to connect to the CPF Web Services. |
| CPF Business Application Plug-ins API | The API that business applications must implement to be exposed as a CPF business application. |
| Business Application Plug-ins | Business applications developed using the CPF Business Application Plug-ins API that can be deployed to the CPF. |

## References

The following documents were used in the creation of this architecture and design:

Table 2 – References

|  |  |  |
| --- | --- | --- |
| Title | Date | Organization |
| SRM Design Standards | 24-Mar-2005 | MAL |

## Overview of Document

This document contains the information and structure required by BC Government Design Standards.

This document contains the following sections:

* **Introduction** – Introduces the purpose and scope of this document.
* **System Overview** – Provides an overview of the project and the components of the system.
* **System Architecture** – Describes the detailed system architecture of the components of the system and components in external systems.
* **Component Design** – Describes the design of the internals of the components of the system
* **Data Design** – Describes the design of the physical data model[[1]](#footnote-1) and any constraints on the implementation of the physical data model.
* **Human Interface Design** – Describes the design of the screen flows, screen layouts and report layouts for the components of the system that weren't covered in the requirements document.
* **Technical Requirements** – Describes any technical requirements that constrain the implementation of the system.
* **Appendices** – The appendices contain additional examples and security matrices used for the implementation of the system.

# System Overview

## Project Perspective

In recent years the Province has been developing web service applications to provide access to spatial data and spatial processing functions. For example the geo-coder web service provides a point location for a given street address. Software applications in addition end users can use web services.

Several issues were identified with the existing web service implementations:

* Many web services were designed to accept a single request at a time. If a user wanted to geo-code 1,000 addresses they would need to submit 1,000 requests to the web service.
* Some web services can take minutes or hours to run. If the connection to the service failed the user would not be able to get the results of the service.
* Web services allow on demand access to the web service function and data, with the only limit to the number of concurrent web server connections on a shared server. This can cause excess load on servers and databases.

In November 2008 the Ministry initiated the Batch Processing Framework (BPF) project to resolve these issues. The BPF project had the following requirements.

* Implement a web service API that had a HTML user interface for end users and a machine-readable (HTML micro-format, XML, or JSON) for software applications.
* Allow users to submit a batch job containing one or more requests to a business application (web service). Batch jobs are executed in the background.
* Allow users to view a list of their jobs and download the results of the jobs.
* Support business applications that take structured data values (e.g. CSV, JSON) or binary data (e.g. JPG, PDF, ESRI Shapefile) as input and result data.
* Provide a simple business application plug-in framework that could be used to develop new web services to be deployed to the CPF.
* Support limits to the number of concurrent executing requests for a business application.
* Support fair scheduling of requests to the business application so that a single user, or single job does not prevent other jobs from being executed.
* Support dynamically adding or removing additional worker servers to handle increased load.
* Support OAuth authentication of web service request.

In January 2010 the second phase of the project was initiated to investigate deploying the BPF framework and workers to the Amazon EC2 cloud environment. This would allow additional worker servers to be added or removed on demand to handle increased load on the system. The BPF was renamed to Cloud Processing Framework (CPF) during this phase.

In December 2010 the third phase of the project was imitated to support the Batch Spatial Overlay Engine implementation for the e-referral project.

* Add support for Siteminder authentication of web services.
* Integrate with WebADE to allow business applications to restrict users access to data and services.
* Support deployment to the Oracle Application Server environment.
* Support delivery of business applications independently of the CPF framework.
* Improve performance with multiple worker servers.
* Support deployment to a private cloud using Ubuntu Enterprise Cloud (Eucalyptus).

## System Context

The Cloud Processing Framework (CPF) is a web service framework to allow end users or other applications submit a job containing one or more requests to a business application, for example 1,000 addresses to a geo-coder business application.

The CPF queues the requests in that, along with requests for jobs from other users for execution. The CPF scheduler will schedule the jobs and requests for execution so that jobs from multiple users get a fair chance of being executed in a timely manner. The CPF can have one or more worker servers to execute the requests from the scheduled jobs. Each worker server has a pool of worker threads, each thread can execute a single request in parallel with the other worker threads. If the worker server has available worker threads it will get the next business application request to execute and start execution in a worker thread.

The CPF supports a business application plug-in API. This API allows additional business applications to be developed and deployed to the CPF application.

A business application can be configured to have a maximum number of requests that can be scheduled at one time. This allows the CPF to limit access to a resource. For example \a business application may only allow 10 concurrent database connections.

## General Constraints

1. The CPF database will be deployed in the shared BMG operational database environment.
2. The CPF database can also be deployed to any Oracle or PostgreSQL database environment.
3. The CPF components will be deployed to the shared OC4J J2EE application server environment.
4. The CPF components can also be deployed to any Tomcat or OC4J 10.1.3 application server environments.
5. The CPF component and database can also be deployed to the Amazon EC2 or an Ubuntu Private Cloud environment.
6. Components deployed outside the Governments network must not be able to access any servers internal to the Government network. All access must be via secured web service interfaces, rather than VPN access.

## Assumptions and Dependencies

1. The CPF application depends on one or more business application plug-ins deployed to the shared plug-in directory.

# System Architecture

The system architecture describes the detailed architecture of the components of the system and components in external systems. The architecture is described using the following five RM-ODP viewpoints that describe a consistent architecture from different perspectives.

* **Enterprise Viewpoint** – Describes the domains, policies and roles of the system.
* **Information Viewpoint** – Describes the information managed and handled by the system and the constraints on the use and interpretation of that information.
* **Computational Viewpoint** – Describes the interfaces used and defined by the components in the system and the interactions between the components using the interfaces.
* **Engineering Viewpoint** – Describes how the components used in the system are distributed across the various servers in the Ministry environment.
* **Technology Viewpoint** – Describes the hardware, software versions and other technologies required by the components of the system.

## Enterprise Viewpoint

The enterprise viewpoint describes the domains, policies and roles of the system.

### Roles

Table 3, shows the roles (actors) within the system and their responsibilities within the scope of the system.

Table 3 – Enterprise Viewpoint Roles

|  |  |
| --- | --- |
| Role Name | Responsibilities |
| Client | * Use the CPF Internal Web Services to submit jobs to the CPF and download the results. |
| User Client | * A Client who is a real person who uses a web browser to access the CPF Client Web Services. |
| Application Client | * A Client that is an application (Java, C#, JavaScript) that uses the CPF Client Web Services. |
| Business Application | * A service that executes a specific function (e.g. geo-coder) generating a response for the given request parameters. |

### Policies

Table 4, shows the key policies for the system.

Table 4 – Enterprise Viewpoint Policies

| Policy Description |
| --- |
| The CPF Client Web Services must support User and Application Clients. |
| The CPF Client Web Services must only allow authenticated users to access secure web services. |
| The CPF Client Web Services must allow OAuth, HTTP Digest and Siteminder authentication. |
| The CPF Client Web Services must only allow Clients to access their own jobs. |
| The CPF Internal Web Services must only allow access to authenticated internal CPF users (e.g. A CPF Worker). |
| The CPF Internal Web Services must allow OAuth and HTTP Digest authentication. |
| The CPF Scheduler must ensure that a CPF Worker executes all requests in a batch job using a business application. |
| The CPF Scheduler must ensure that if a CPF Worker stops responding that the requests are distributed to other CPF Workers for execution. |
| The CPF Scheduler should implement a fair scheduling algorithm so that a single user or job does not prevent other jobs from being executed. |
| The CPF Scheduler must remove jobs 7 days after the job was completed. |
| The CPF must allow business applications to enforce access control permissions using WebADE action permissions and WebADE roles. |

## Information Viewpoint

The information viewpoint describes the information managed and handled by the system and the constraints on the use and interpretation of that information.

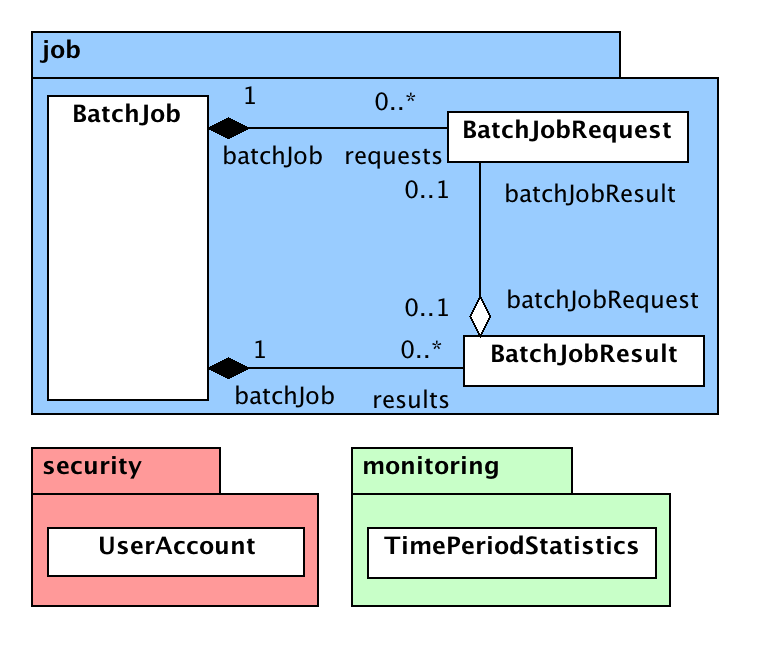


Figure 1 – CPF Information Viewpoint

Table 5 – CPF Class Descriptions

|  |  |
| --- | --- |
| Class | Description |
| BatchJob | This class represents the definition of a job |
| BatchJobRequest | This class represents the structured data parameters or input opaque data for each request in a job to be executed by a business application. The class also stores the status of the request and |
| BatchJobResult | This class represents a result file generated after processing all the requests in a job. For structured data there will be one file, and optionally one error file. For opaque data there will be one file per successful request, and optionally one error file. |
| UserAccount | This class represents an internal CPF user or an external user (e.g. IDIR, BCeID user). Each user account is given a consumer key and consumer secret that can be used to login to the web services using OAuth or HTTP Digest. |
| TimePeriodStatistic | This class represents a numerical statistic collected over a time period. For example number of jobs by business application for a day or hour. |

## Computational Viewpoint

The computational viewpoint describes the interfaces used and defined by the components in the system and the interactions between the components using the interfaces.

The following diagram shows the parallel processes (threads) used by the CPF and the communication channels between these processes. Web services are shown as blue ovals. Multiple client requests to these web services can be handled concurrently in threads managed by the application server. The green boxes are a single instance of a thread managed by the CPF. These threads listen on one or more communications channels and perform a task based on the message sent over the thread. The grey triangle within a green box indicates that the process listens to more than one channel. Communications channels allow more than one message to be queued on the channel, the name of these queues are shown in the orange boxes. Both the scheduler server and worker server use thread pools to execute certain tasks. These tasks (runnable classes) shown in red boxes are scheduled by another process to be executed in one of the pool of parallel threads. The thread pool can be used to execute pre-process, job request schedule or post-process tasks on the scheduler server or invoke a business application plug-in on the worker server.



Figure 2 - Parallel Processing Diagram

New jobs are submitted via the submit single or multiple requests web services. If the job was submitted via the multiple requests interface using structured data the job identifier is sent to the Job Pre-Process Queue. Otherwise it is sent to the Job-Schedule Queue.

The Job Pre-Process Scheduler Thread on start-up loads any jobs that have not been pre-processed from the database and sends a Pre-Process runnable for the job to the Job Pre-Process Run Queue. It then has an infinite loop that reads from the Job Pre-Process Queue and sends a Pre-Process runnable for the job to the Job Pre-Process Run Queue.

The Pre-Process Runnable is executed in a thread from the Scheduler Task Thread Pool when one becomes available. It reads the structured input data for a job and creates a batch job request from each record in the input data. When all records have been processed it sends the job identifier to the Job Schedule Queue.

The Job Request Scheduler Thread on start-up loads any jobs that have requests to be scheduled from the database and sends a Request Schedule runnable for the job to the Job Request Schedule Queue. It then has an infinite loop that reads from the App Busy Job Schedule Queue or the Job Schedule Queue. Preference is give to the App Busy Queue and sends a Request Schedule runnable for the job to the Job Request Schedule Queue.

The Request Schedule Runnable reads BusinessApplicationPlugin.numRequestsPerWorker requests that have not been scheduled for the job from the database. If there are no requests to process no action is performed. If the number of scheduled job request groups for business application exceeds BusinessApplicationPlugin. maxConcurrentRequests the job identifier will be sent to the App Busy Job Queue to be re-scheduled when the business application is not busy. If there were requests and the business application is not busy a new batch job request execution group is created for the requests and sent to the Job Request Group Scheduler Queue. The batch job identifier is then sent to the back of the Job Schedule Queue so that any additional requests in the job will be scheduled when other jobs have had a chance to be scheduled.

Each worker server has a Worker Scheduler Thread with an infinite loop. The loop requests the identifier of the next batch job request execution group using the web service. It can specify the list of business applications that the worker supports. The web service reads from the Job Request Group Scheduler Queue. If no groups are available for the lost of business application the web service waits for up to 60 seconds for a request to become available, this avoids excessive polling of the server. If a group was available the worker scheduler thread creates a new Plugin Executor runnable for the group and schedule this for execution in a thread in the thread pool. If there are no available threads in the pool the Worker Scheduler Thread invokes the Worker Ping request on the web service to indicate that it is still alive but is not accepting any new work.

The plugin executor runnable gets the parameters for the requests in the group via a web service (not shown on the diagram). It then loops through each of the requests in the group and creates a new instance of the plugin for each request and invokes the plugin. When all requests have been processed in the group it invokes the Job Request Group Results Web Service with the results for all the requests. If the input data or result data are opaque binary blobs then the input and output stream is connected to a web service on the scheduler to read or update the request and result data.

The Job Request Group Results Web Service updates the requests with the results from the group results sent from the worker. If there were queued jobs in the App Busy Queue and the business application is no longer busy the jobs for that business application are sent to the App Busy Job Schedule Queue so that they will be re-scheduled. If all the requests for a business application have been processed then the batch job identifier is sent to the Job Post-Process Queue.

The Job Post-Process Scheduler Thread on start-up loads any jobs that have not been post-processed from the database and sends a Post-Process runnable for the job to the Job Post-Process Run Queue. It then has an infinite loop that reads from the Job Post-Process Queue and sends a Post-Process runnable for the job to the Job Post-Process Run Queue.

The Post-Process Runnable is executed in a thread from the Scheduler Task Thread Pool when one becomes available. It creates the batch job results entries for the structured result data and if there were errors an error result file. It then notifies the user if a notification URL was specified.

The following diagram shows jobs, groups and results for two users. The numbers indicate the order in which the objects are created. The diagram shows that if all 4 jobs are submitted before the first group is scheduled that one group is scheduled for each job in sequence. When all jobs have had a group scheduled then the first job will have the next group scheduled.



Figure 3 – Job Group Decomposition

The following diagram shows the queuing of jobs and execution groups and the parallel execution of tasks. Each column shows the list of tasks that have been executed (top of list), those that are currently executing in parallel (red box) and those queued to be executed (bottom of list). The job Schedule column shows that the jobs are repeatedly sent to the back of the Job Schedule queue after scheduling until all requests have been scheduled.



Figure 4 - Parallel Task Execution

### Component Interactions

The following section describes the main interactions between the actors and components of the system. The interactions are described using UML sequence diagrams and descriptive text. The goal of these interactions is to give an overview of the steps not to be an exact representation of the code.

#### Client Job Submission

The following sequence diagram shows the typical sequence of CPF client web service calls a Client makes to submit a new batch job and download the results of the job.

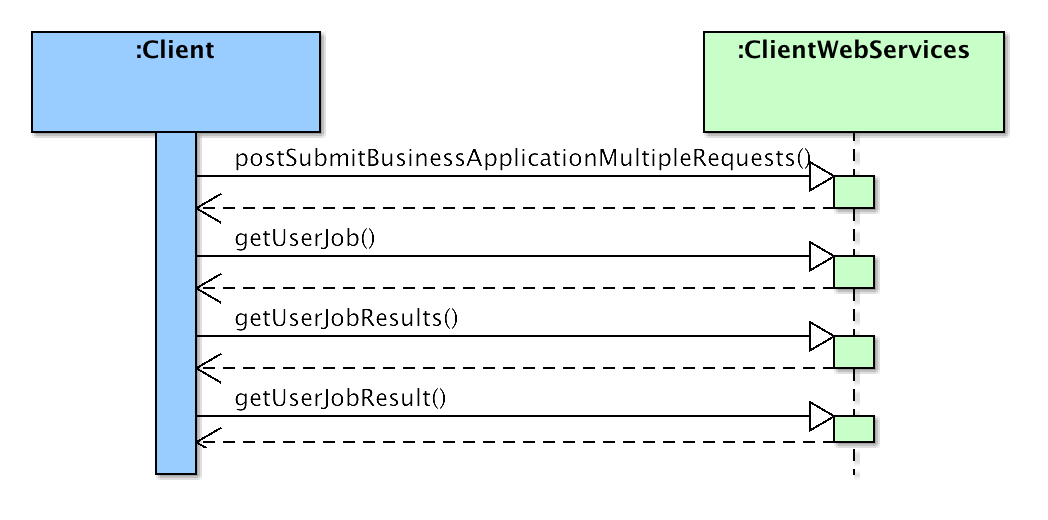


Figure 5 – Client Job Submission Sequence Diagram

The client submits a new job by issuing a POST request to the Submit Business Application (Single or Multiple) Requests web service. This creates a new batch job for the requests and redirects the user to the User Job web service for the created batch job.

The user issues a GET request to the User Job web service to get the status of the job. The user repeats this periodically until the job is completed; at this point the User Job page includes a URL to the User Job Results resource. The job status indicates the frequency that the user should poll the User Job web service.

The user issues a GET request to the User Job Results web service to get a list of the URLs to User Job Results resources for the batch job.

The user issues a GET request to the User Job Result web service to download the result file for each of the URLs returned from the User Job Results web services.

#### Batch Job State Transitions

The following diagram shows the state transitions for a batch job.

If a batch job is created using the Submit Business Application Single Request web service or if the Business Application accepts per request input data the Submitted and Creating Requests states are skipped and the batch job state will be set to Requests Created.

If there was an error in the Creating Requests phase an error result file will be created and the state changed to Results Created.

If the CPF was restarted any batch jobs in the Creating Requests state will be reset to the Submitted state so that the requests will be created.

If the CPF was restarted any batch jobs in the Creating Results state will be reset to the Processed state so that the results will be created.

Any batch jobs in the Results Created or Download Initiated state can be cleaned up up after 7 days. The final state transition is performed once the batch job has been deleted from the system.

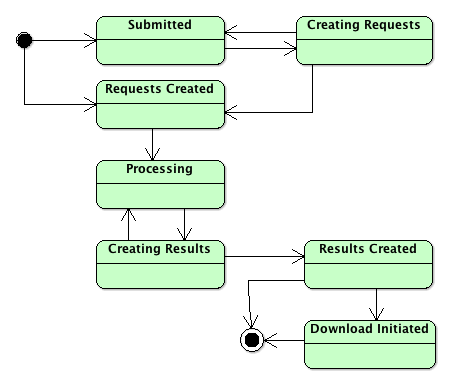


Figure 6 – Batch Job State Diagram

#### Executor Service

The ExecutorService is used to execute tasks in a separate thread. This allows complex tasks and database interactions to occur in separate threads.

The service maintains a pool of threads that can be used to execute tasks. The size of this pool is used to control how much concurrent tasks are executed. On a more powerful machine the thread pool size can be increased to improve the throughput of the system.

The ExecutorService reads a task to execute from one the output channels of the pre-processor, scheduler and post-processor. If there is an idle thread in the pool of workers the task is executed in one of the threads, otherwise the task is queued until a thread is available.

If there is a post-process task to execute it takes precedence over the scheduler tasks and the scheduler tasks take precedence over the pre-processor tasks.

#### Batch Job Pre-Processor

The batch job pre-processor is used to create the BatchJobRequest records for business applications that use structured input data.

The pre-process is initiated by the Client Web Services sending the batch job id to pre-processors input channel. This queues the job to be handled by the pre-processor.

The pre-process reads the next job to pre-process from the input channel. A new task is created to execute the preProcessBatchJob method on the BatchJobService. This task is sent to the ExecutorService for execution.

The ExecutorService executes the preProcessBatchJob method on the BatchJobService using a thread from the pool.

Before processing the batch job status is updated to Creating Requests. The pre-processor reads the structured input data from the BatchJob inputData or inputDataUrl. A BatchJobRequest record is created for each record in the input data file. When all records are processed the batch job status is changed to Requests Created. The Scheduler is then notified that there is a new batch job to be scheduled.

If there are no jobs to pre-process after 60 seconds the pre-processor checks the database to see if there were any that were missed.

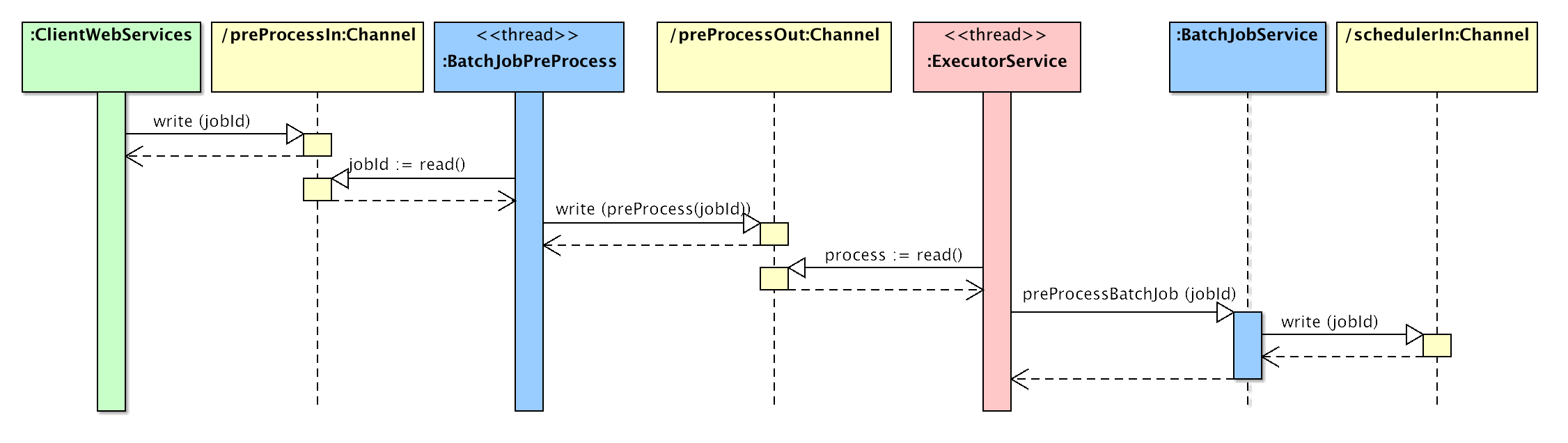


Figure 7 Batch Job Pre-Process Sequence Diagram

#### Batch Job Scheduler

The batch job scheduler is used to schedule groups of BatchJobRequests from a BatchJob for execution by the CPF Workers.

The Client Web Services, Pre-Processor, or the Scheduler initiates the scheduler for a batch job by sending the batch job id to the scheduler input channel. This queues the job to be handled by the scheduler.

The scheduler reads the next job to schedule from the queued input channel or if none available from the input channel. A new task is created to execute the scheduleBatchJobRequests method on the batch job service. This task is sent to the ExecutorService for execution. The ExecutorService executes the scheduleBatchJobRequests method on the BatchJobService using a thread from the pool.

If the batch job status is requests created it is updated to processing. The batch job requests are queried for the batch job. If there are no requests to process then the job will not be re-scheduled as the requests are completed one or more workers are currently executing all requests. Next the number of concurrent requests for a business application is compared with the maximum for that business application.

* If the limit has been exceeded the job will be queued to be scheduled when the limit is not exceeded.
* Otherwise a sequential execution group is created for the batch job requests. The requests are marked as started so they will not be scheduled again. The execution group is added to the list of tasks to be executed by the CPF workers. Finally the batch job id is sent to the scheduler input channel so that any additional requests will be scheduled.

Queued jobs will be send to the scheduler queued input channel when execution groups for that business application are completed. These will take precedence over any new jobs to schedule.

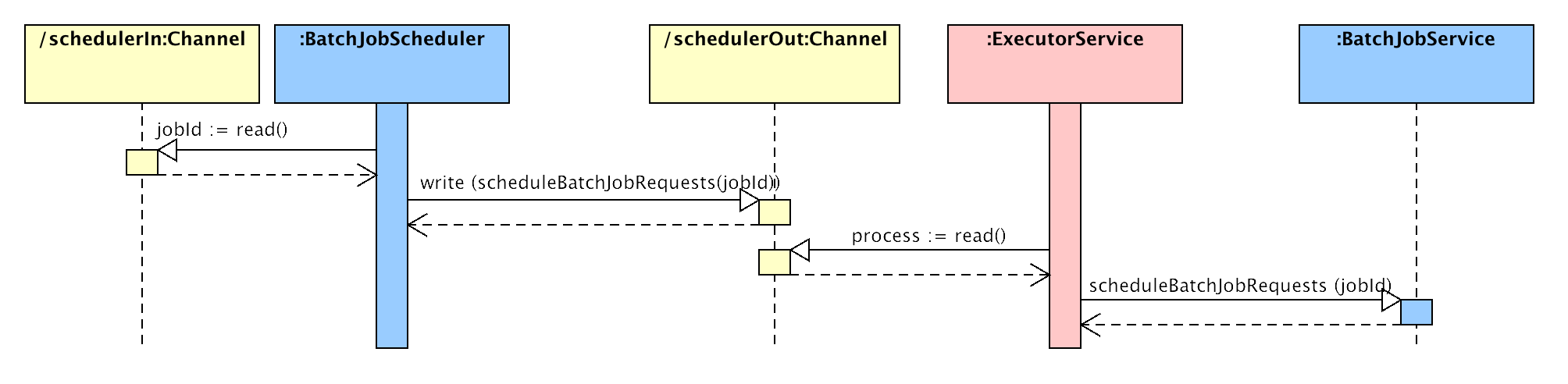


Figure 8 – Batch Job Scheduler Sequence Diagram

#### Batch Job Post-Processor

The batch job post-processor is used to create the BatchJobResult records for business applications that use structured input data and the BatchJobResult record for any requests that generated an error.

The post-process is initiated when all requests for the batch job have been processed. The batch job service sends the batch job id to post-processors input channel. This queues the job to be handled by the post-processor.

The post-process reads the next job to post-process from the input channel. A new task is created to execute the postProcessBatchJob method on the batch job service. This task is sent to the ExecutorService for execution.

The ExecutorService executes the postProcessBatchJob method on the BatchJobService using a thread from the pool.

Before processing the batch job status is updated to Creating Results. The pre-processor invokes the createStructuredResults method on the BatchJobService. The structured result data BatchJobResult file is created by adding one record for the structured resultData from each BatchJobRequest that was successful. The error BatchJobResult file is created by adding one record for the error details from each BatchJobRequest that failed. The batch job status is updated to Results Created. If a notification URL was specified the client is notified by a URL callback or email that the results have been created.

If there are no jobs to post-process after 60 seconds the pre-processor checks the database to see if there were any that were missed.

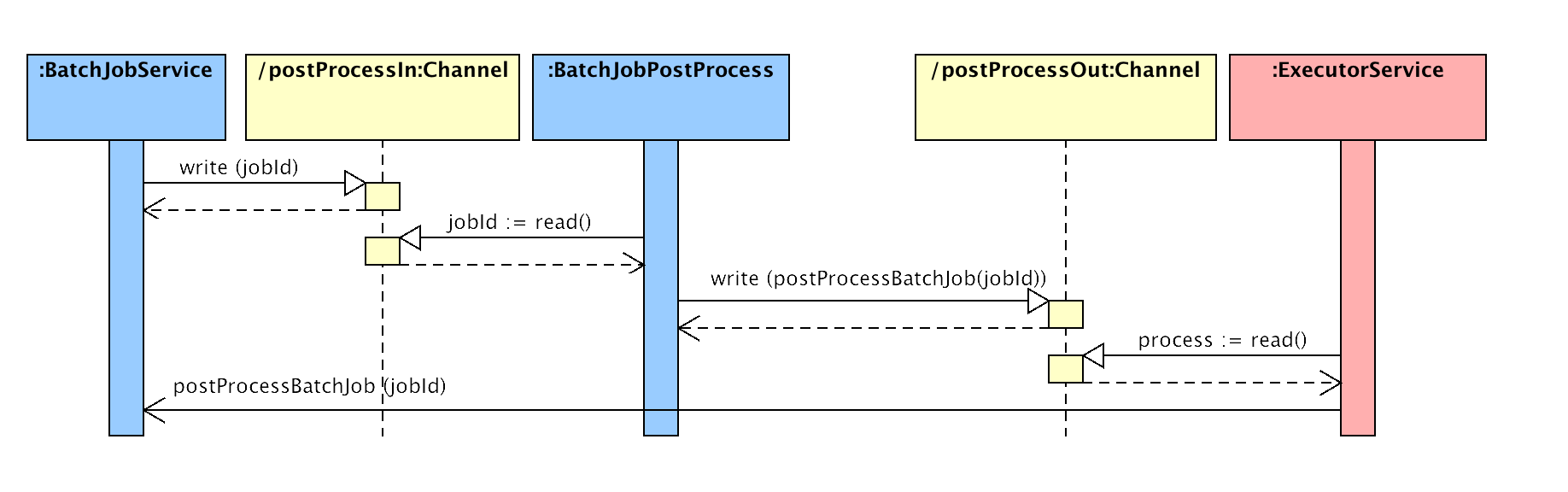


Figure 9 Batch Job Post-Process Sequence Diagram

#### Worker Batch Job Request Group Execution

The CPF worker executes the business application for each batch job request in a batch job request execution group. The worker maintains a pool of threads to execute the business application requests.

The BatchJobWorkerExecutor is a thread that polls the postNextBatchJobExecutionGroupId internal web service to get the next ID of a job to process for a list of business applications. Different worker instances can be configured to process either all or a subset of the available business applications. If there is not a request to execute the internal web services will wait for 60 seconds for a new request to be scheduled. This reduces the number of round trips to the server but allows the worker to issue a new request every 60 seconds to indicate that it is still alive. If the worker does not have any available threads to process requests then it can issue the postWorkerPing internal web service to indicate that it is still alive.

If there was a batch job request group to execute and there is a thread available the BatchJobWorkerExecutor runs the BatchJobWorkerRunner in a pooled thread. The BatchJobWorkerRunner invokes the getBatchJobExecutionGroup on the InternalWebServices to get the details of the requests to execute for that group. The runner loops through each of the requests specified in the group. If the business application accepts per request input data the getBatchJobExecutionGroup on the InternalWebServicesis invoked and an input stream passed to the business application plugin.

The worker sets the parameters on the plug-in and the execute method is invoked on the BusinessApplicationPlugin for each batch job request to perform the work of the request. If the business application returns opaque result data then the postBatchJobExecutionGroupResults internal web service is invoked when the plug-in attempts to write the output. When the execution has finished the results or error is added to the group results object.

When all requests in the group have been completed the postBatchJobExecutionGroup internal web service is invoked with the group results object. The internal web service will record the results or error of each request in the database. The post-processor will be notified to process the job if all requests for the job have been completed.

If the plug-in uses the CPF access control mechanisms then it will be returned a caching web service enabled security service for the user associated with the job. This security service will use cached security permissions if available, otherwise it will invoke the security internal web services to get the user’s attributes or to check if they have the permission being requested. Cached information is kept for 15 minutes.

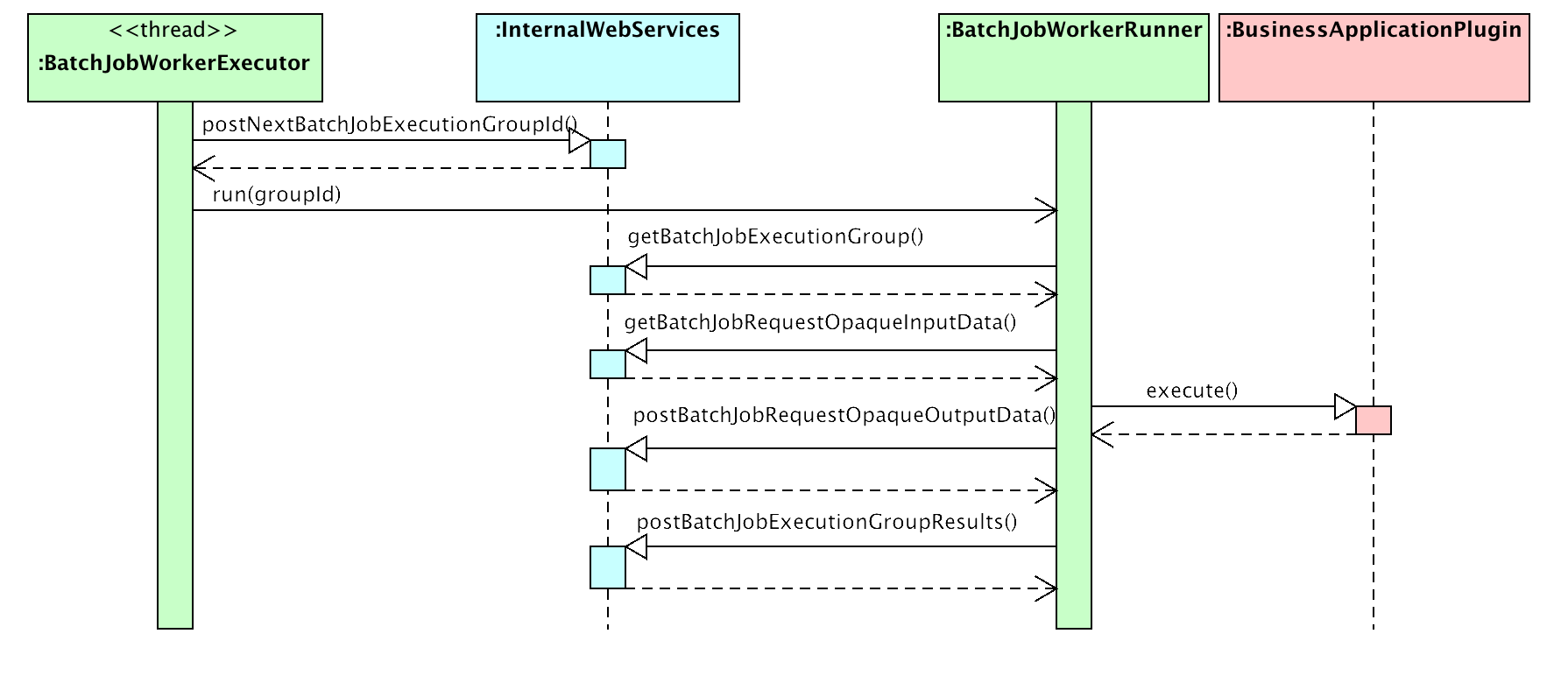


Figure 10 – Worker Batch Job Request Group Execution

### Interface Specifications

The CPF uses HTTP based web services to expose the externally accessible interfaces. The term resource is used to describe a web service interface in this document. There are two groups of web services client and internal. The client web services are publically available to any CPF client. The client web services provide interfaces to submit jobs to the CPF and check on the status of jobs. The internal web services are restricted to CPF worker user accounts. The internal web services are used to distribute requests and the results of those requests to the CPF worker instances.

Each resource has a path from the root of the CPF application, for example the path /ws/users would be deployed to <http://apps.gov.bc.ca/pub/cpf/ws/users>. Most resources require login using OAuth or HTTP digest. Login using Siteminder and the common login page is supported by prefixing the path with /secure, for example <http://apps.gov.bc.ca/pub/cpf/secure/ws/users>. For Siteminder the user will be redirected to common login page, this means that this version of the web services can only be used by end users using a web browser and cannot be accessed by applications developed using Java or C#. The path can be specified with or without the trailing /, for example /ws/users and /ws/users/ are both valid.

HTML is returned as the default representation for a resource. A different representation can be requested using the HTTP Accept header[[2]](#footnote-2), format query string parameter or appending the file extension to the path.

* The HTTP Accept header allows a list of the media types the client will accept in order of preference.
* The format query string parameter can either contain a single media type or the file extension for that media type.
* The file extension can be appended to the path, separated by a period, for example /ws/users.xml. No trailing / must be added to the path in this case, for example /ws/users/.xml **is not valid.**

The representation can also be selected or by adding a file extension to the end of the path. Table 6 lists all the supported media types and their file extensions. The list of the supported Content Types is included in the description of each web service below.

Table 6 – Media Types and file extensions

| Media Type | File Extension | Description |
| --- | --- | --- |
| text/html | html | (X)HTML styled representation of resource. |
| text/xml | xml | * XML list describing the child resources below this resource. * Simple XML representation of the resource value. |
| application/json | json | * JSON[[3]](#footnote-3) list describing the child resources below this resource (see 3.3.2.1.1.2). * Simple JSON object representation of the resource value. |
| application/vnd.sun.wadl+xml | wadl | Web Application Description Language[[4]](#footnote-4) XML document describing the child resources below this resource. |
| text/uri-list | uri-list | Simple text document with one URL to each child resource below this resource. |

#### Resource Representations

The following section describes the various types of representation generated by the CPF web service resources.

##### Resource List

The Resource List representations below are used when a resource returns a list of child resources.

###### HTML Resource List

The HTML representation is a formatted HTML page that uses a definition list containing the list of resources. The definition list is wrapped in a div with class resources. The dt for each resource contains an anchor tag with the URI to the resource and displays the resource title. The dd for each resource contains the resource description.

By Default the HTML page contains a BC Government header, navigation and footer. This can be turned off using the plain=true query string parameter.

<div class="resources">

<dl>

<dt><a href="**Child Resource URL**">**Child Resource Title**</a></dt>

<dd><p>**Child Resource Description**.</p></dd>

<dt><a href="**http://apps.gov.bc.ca/pub/cpf/ws/apps/**">**Business Applications**</a></dt>

<dd><p> **List of business applications.**</p></dd>

</dl>

</div>

###### JSON Resource List

The JSON representation is an object containing the items attribute containing a list of resource objects. The resource object contains the resourceUri, title and description attributes.

{

"items": [

{

"resourceUri" : "**Child Resource URL**",

"title" : "**Child Resource Title**",

"description" : "**Child Resource Description**"

},

{

"resourceUri" : "**http:\/\/apps.gov.bc.ca\/pub\/cpf\/ws\/apps\/**",

"title" : "**Business Applications**",

"description" : "**List of business applications.**"

}

]

}

###### XML Resource List

The XML representation has the items root element with zero or more resource elements. The resource element contains the resourceUri, title and description elements.

<?xml version="1.0" encoding="UTF-8"?>

<items>

<resource>

<resourceUri>**Child Resource URL**</resourceUri>

<title>**Child Resource Title**</title>

<description>**Child Resource Description**</description>

</resource>

<resource>

<resourceUri>**http://apps.gov.bc.ca/pub/cpf/ws/apps/**</resourceUri>

<title>**Business Applications**</title>

<description>**List of business applications.**</description>

</resource>

</items>

###### WADL Resource List

The WADL representation contains a root resource for the current resource, with a list of child resources.

<?xml version="1.0"?>

<wadl:application xmlns:wadl="http://research.sun.com/wadl/2006/10">

<wadl:resources>

<wadl:resource path="**Resource URL**">

<wadl:doc title="**Resource Title** ">**Resource Description** </wadl:doc>

<wadl:resource path="**Child Resource URL**">

<wadl:doc title="**Child Resource Title**">**Child resource Description**

</wadl:doc>

</wadl:resource>

<wadl:resource path="**http://apps.gov.bc.ca/pub/cpf/ws/apps/**">

<wadl:doc title="**Business Applications List**">**List of business**

**applications.**</wadl:doc>

</wadl:resource>

</wadl:resource>

</wadl:resources>

</wadl:application>

###### URI-List Resource List

The uri-list representation is a text file with a URL to each of the child resources on a separate line.

**Child Resource URL**

**http://apps.gov.bc.ca/pub/cpf/ws/apps/**

##### Object Detail

The object detail representations below are used when a resource returns the values of an object.

###### HTML Object Detail

The HTML representation is a formatted HTML page that uses a table with two columns containing the object attribute names & values. The table is wrapped in a div with class objectView. The table has the class data. Each attribute is represented as a table row. The attribute name is wrapped in a th element and the value in a td. Any URL values are displayed as an anchor element.

By Default the HTML page contains a BC Government header, navigation and footer. This can be turned off using the plain=true query string parameter.

The following shows the general syntax of the HTML object detail representation.

<div

class="objectView"

>

<table

class="data"

>

<tbody>

<tr>

<th>**Attribute 1**</th>

<td>

<a

href="**URL Value**"

>**URL Value**</a>

</td>

</tr>

<tr>

<th>**Attribute 2**</th>

<td>**Value 2**</td>

</tr>

</tbody>

</table>

</div>

The following example shows an XML object detail representation of a BatchJob.

<div class="objectView">

<table class="data">

<tbody>

<tr>

<th>Id</th>

<td>

<a

href="http://delivery.apps.gov.bc.ca/pub/cpf/ws/users/cpftest/jobs/25/"

>http://delivery.apps.gov.bc.ca/pub/cpf/ws/users/cpftest/jobs/25/</a>

</td>

</tr>

<tr>

<th>User Id</th>

<td>cpftest</td>

</tr>

<tr>

<th>Business Application Name</th>

<td>MapTileByTileId</td>

</tr>

<tr>

<th>Business Application Version</th>

<td>1.0.0</td>

</tr>

<tr>

<th>Map Grid Name</th>

<td>NTS 1:1 000 000</td>

</tr>

<tr>

<th>Job Status</th>

<td>downloadInitiated</td>

</tr>

<tr>

<th>Milliseconds Until Next Check</th>

<td>0</td>

</tr>

<tr>

<th>Num Submitted Requests</th>

<td>1</td>

</tr>

<tr>

<th>Num Completed Requests</th>

<td>1</td>

</tr>

<tr>

<th>Num Failed Requests</th>

<td>0</td>

</tr>

<tr>

<th>Result Data Content Type</th>

<td>application/json</td>

</tr>

<tr>

<th>Results Url</th>

<td>

<a

href="http://delivery.apps.gov.bc.ca/pub/cpf/ws/users/cpftest/jobs/25/results"

>http://delivery.apps.gov.bc.ca/pub/cpf/ws/users/cpftest/jobs/25/results</a>

</td>

</tr>

</tbody>

</table>

</div>

###### JSON Object Detail

The JSON representation is an object with one child attribute containing the attribute value for each non-null attribute of the object.

The following shows the general syntax of the XML object detail representation.

{

"**attribute1**": "**Value 1**",

"**attribute2**": "**Value 2**"

}

The following example shows an XML object detail representation of a BatchJob.

{

"id":"\/cpf\/ws\/users\/cpftest\/jobs\/25.json",

"userId":"cpftest",

"businessApplicationName":"MapTileByTileId",

"businessApplicationVersion":"1.0.0",

"mapGridName":"NTS 1:1 000 000",

"jobStatus":"downloadInitiated",

"millisecondsUntilNextCheck":"0",

"numSubmittedRequests":"1",

"numCompletedRequests":"1",

"numFailedRequests":"0",

"resultDataContentType":"application\/json",

"resultsUrl":" http:\/\/apps.gov.bc.ca\/pub/\/cpf\/ws\/users\/cpftest\/jobs\/25\/results"

}

###### XML Object Detail

The XML representation has a root element with the type name of the object (e.g. BatchJob). The root element has one child element containing the attribute value for each non-null attribute of the object.

The following shows the general syntax of the XML object detail representation.

<?xml version="1.0" encoding="UTF-8"?>

<**TypeName**>

<**attribute1**>**Value 1**</**attribute1**>

<**attribute2**>**Value 2**</**attribute2**>

</**TypeName**>

The following example shows an XML object detail representation of a BatchJob.

<?xml version="1.0" encoding="UTF-8"?>

<BatchJob>

<id>/cpf/ws/users/cpftest/jobs/25.xml</id>

<userId>cpftest</userId>

<businessApplicationName>MapTileByTileId</businessApplicationName>

<businessApplicationVersion>1.0.0</businessApplicationVersion>

<jobStatus>downloadInitiated</jobStatus>

<millisecondsUntilNextCheck>0</millisecondsUntilNextCheck>

<numSubmittedRequests>1</numSubmittedRequests>

<numCompletedRequests>1</numCompletedRequests>

<numFailedRequests>0</numFailedRequests>

<resultDataContentType>application/json</resultDataContentType>

<resultsUrl>http://apps.gov.bc.ca/pub/pub/cpf/ws/users/cpftest/jobs/25/results</resultsUrl>

</BatchJob>

#### Client Web Services

The client web services are publically available to any CPF client. The client web services provide interfaces to submit jobs to the CPF and check on the status of jobs.

##### CPF Client Root

The CPF client root resource is the top-level entry point for the CPF client web services. The representation returns links to the Business Applications and Users resources.

Note that this resource has the paths /ws/ and /ws/index. The path /ws/ cannot be used for representations using the file name extension (e.g. /ws.json is not valid). In these cases the path /ws/index (e.g. /ws/index.json) can be used.

|  |  |
| --- | --- |
| Path | /ws/,/ws/index |
| Method | GET |
| Content Types | text/html, text/xml, application/json, application/vnd.sun.wadl+xml, text/uri-list |
| Representation | Resource list |

##### Business Applications

The business applications resource is a container for the business applications deployed to the CPF. The representation returns links to the Business Application Version resources for the latest version of each business application.

|  |  |
| --- | --- |
| Path | /ws/apps |
| Method | GET |
| Content Types | text/html, text/xml, application/json, application/vnd.sun.wadl+xml, text/uri-list |
| Representation | Resource list |

##### Business Application

The business application resource is a container for the versions of a business application. The representation returns links to the Business Application Version resources for all supported versions of a business application.

|  |  |
| --- | --- |
| Path | /ws/apps/{businessApplicationName} |
| Method | GET |
| Content Types | text/html, text/xml, application/json, application/vnd.sun.wadl+xml, text/uri-list |
| Representation | Resource list |

Figure 11 – Business Application Parameters

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Parameter Type | Data Type | Description |
| businessApplicationName | Path Variable | String | The name of the business application. |

##### Business Application Version

The business application version resource is a container for the version of a business application. The representation returns links to the Business Application Multiple Requests Form and Business Application Single Request Form resources.

|  |  |
| --- | --- |
| Path | /ws/apps/{businessApplicationName}/{businessApplicationVersion} |
| Method | GET |
| Content Types | text/html, text/xml, application/json, application/vnd.sun.wadl+xml, text/uri-list |
| Representation | Resource list |

Figure 12 – Business Application Version Parameters

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Parameter Type | Data Type | Description |
| businessApplicationName | Path Variable | String | The name of the business application. |
| businessApplicationVersion | Path Variable | String | The version of the business application. |

##### Business Application Multiple Requests Form

The business application multiple requests for resource is a HTML form that can be used to submit a new batch job containing multiple business application requests to the Submit Business Application Multiple Requests resource. The representation is a HTML form that contains the fields to upload the job parameters, input data and business application specific job parameters for the batch job.

|  |  |
| --- | --- |
| Path | /ws/apps/{businessApplicationName}/{businessApplicationVersion}/multiple |
| Method | GET |
| Content Types | text/html |
| Representation | HTML Form |

Figure 13 – Business Application Multiple Requests Form Parameters

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Parameter Type | Data Type | Description |
| businessApplicationName | Path Variable | String | The name of the business application. |
| businessApplicationVersion | Path Variable | String | The version of the business application. |

##### Submit Business Application Multiple Requests

The submit business application multiple requests resource is a HTTP POST request that creates a new job containing multiple business application requests from the input parameters.

|  |  |
| --- | --- |
| Path | /ws/apps/{businessApplicationName}/{businessApplicationVersion}/multiple |
| Method | POST |
| Content Types | text/html, text/xml, application/json |
| Request Content Types | multipart/form-data |
| Representation | HTTP 303 See Other GET redirect to User Job resource for the created batch job. |

Figure 14 – Business Application Multiple Requests Form Parameters

| Name | Parameter Type | Data Type | Description |
| --- | --- | --- | --- |
| businessApplicationName | Path Variable | String | The name of the business application. |
| businessApplicationVersion | Path Variable | String | The version of the business application. |
| numRequests | Form Field | Integer | The optional number of business application requests in the job. |
| inputDataContentType | Form Field | String | The mime media type of the inputDataUrl or inputData. Can be repeated if the media type changes for each input data. |
| inputDataUrl | Form Field | String | The URL to the input data. For structured input data this must only be specified once. For opaque input data this must be repeated with the data for each business application request. Mutually exclusive with inputData. |
| inputData | Form Field | Binary | The input data. For structured input data this must only be specified once. For opaque input data this must be repeated with the data for each business application request. Mutually exclusive with inputDataUrl. |
| resultDataContentType | Form Field | String | The mime media type to return the result data using. |
| notificationUrl | Form Field | String | The http(s) or mailto: URL used to notify the client when the job has been completed. |
| <app job parameters> | Form Field | Varies | Any business application job parameters can be specified as form fields. |

##### Business Application Single Request Form

The business application single request for resource is a HTML form that can be used to submit a new batch job containing a single business application request to the Submit Business Application Single Requests resource. The representation is a HTML form that contains the fields to specify the job parameters and business application job and request parameters for the business application request.

|  |  |
| --- | --- |
| Path | /ws/apps/{businessApplicationName}/{businessApplicationVersion}/single |
| Method | GET |
| Content Types | text/html |
| Representation | HTTP 303 See Other GET redirect to User Job resource for the created batch job. |

Figure 15 – Business Application Single Request Form Parameters

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Parameter Type | Data Type | Description |
| businessApplicationName | Path Variable | String | The name of the business application. |
| businessApplicationVersion | Path Variable | String | The version of the business application. |

##### Submit Business Application Single Request

The submit business application single requests resource is a HTTP POST request that creates a new job containing a single business application request from the input parameters.

|  |  |
| --- | --- |
| Path | /ws/apps/{businessApplicationName}/{businessApplicationVersion}/multiple/ |
| Method | POST |
| Content Types | text/html |
| Request Body Content Types | multipart/form-data, application/form-url-encoded |

Figure 16 – Business Application Multiple Requests Form Parameters

| Name | Parameter Type | Data Type | Description |
| --- | --- | --- | --- |
| businessApplicationName | Path Variable | String | The name of the business application. |
| businessApplicationVersion | Path Variable | String | The version of the business application. |
| inputDataContentType | Form Field | String | The mime media type of the inputDataUrl or inputData. For opaque input data only. |
| inputDataUrl | Form Field | String | The URL to the input data. For opaque input data only. Mutually exclusive with inputData. |
| inputData | Form Field | Binary | The input data. For opaque input data only. Mutually exclusive with inputDataUrl. |
| resultDataContentType | Form Field | String | The mime media type to return the result data using. |
| notificationUrl | Form Field | String | The http(s) or mailto: URL used to notify the client when the job has been completed. |
| <app job parameters> | Form Field | Varies | Any business application job parameters can be specified as form fields. |
| <app request parameters> | Form Field | Varies | Any business application request parameters can be specified as form fields. For structured input data only. |

##### Users

The users resource is a container for the users in the system. The representation returns links to the user resource for the current user. The user can only see their user resource.

|  |  |
| --- | --- |
| Path | /ws/users |
| Method | GET |
| Content Types | text/html, text/xml, application/json, application/vnd.sun.wadl+xml, text/uri-list |
| Representation | Resource list |

##### User

The users resource is a container for the resources associated with a user. The representation returns links to the user applications and user jobs resources. The user can only see their user resource and resources below that resource.

|  |  |
| --- | --- |
| Path | /ws/users/{userId} |
| Method | GET |
| Content Types | text/html, text/xml, application/json, application/vnd.sun.wadl+xml, text/uri-list |
| Representation | Resource list |

Figure 17 – User Parameters

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Parameter Type | Data Type | Description |
| userId | Path Variable | String | The consumer key of the user. |

##### User Applications

The user applications resource is a container for the business applications resources associated with a user. The representation returns links to the user application resource.

|  |  |
| --- | --- |
| Path | /ws/users/{userId}/apps |
| Method | GET |
| Content Types | text/html, text/xml, application/json, application/vnd.sun.wadl+xml, text/uri-list |
| Representation | Resource list |

Figure 18 – User Applications Parameters

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Parameter Type | Data Type | Description |
| userId | Path Variable | String | The consumer key of the user. |

##### User Application

The user application resource is a container for the business application resources associated with a user. The representation returns links to the user application jobs resource.

|  |  |
| --- | --- |
| Path | /ws/users/{userId}/apps/{businessApplicationName} |
| Method | GET |
| Content Types | text/html, text/xml, application/json, application/vnd.sun.wadl+xml, text/uri-list |
| Representation | Resource list |

Figure 19 – User Application Parameters

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Parameter Type | Data Type | Description |
| userId | Path Variable | String | The consumer key of the user. |
| businessApplicationName | Path Variable | String | The name of the business application. |

##### User Application Jobs

The user application jobs resource is a container for the user’s jobs for the business application. The representation returns links to the user job resource for each of the jobs.

|  |  |
| --- | --- |
| Path | /ws/users/{userId}/apps/{businessApplicationName}/jobs |
| Method | GET |
| Content Types | text/html, text/xml, application/json, application/vnd.sun.wadl+xml, text/uri-list |
| Representation | Resource list |

Figure 20 – User Application Jobs Parameters

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Parameter Type | Data Type | Description |
| userId | Path Variable | String | The consumer key of the user. |
| businessApplicationName | Path Variable | String | The name of the business application. |

##### User Jobs

The user jobs resource is a container for the user’s jobs. The representation returns links to the user job resource for each of the jobs.

|  |  |
| --- | --- |
| Path | /ws/users/{userId}/jobs |
| Method | GET |
| Content Types | text/html, text/xml, application/json, application/vnd.sun.wadl+xml, text/uri-list |
| Representation | Resource list |

Figure 21 – User Jobs Parameters

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Parameter Type | Data Type | Description |
| userId | Path Variable | String | The consumer key of the user. |

##### User Job

The user job resource displays the details of a job and the status of the job. If the job had been completed a link to the user job results resource will be returned.

|  |  |
| --- | --- |
| Path | /ws/users/{userId}/jobs/{batchJobId} |
| Method | GET |
| Content Types | text/html, text/xml, application/json, text/uri-list |
| Representation | Object Detail for a BatchJob |

Figure 22 – User Job Parameters

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Parameter Type | Data Type | Description |
| userId | Path Variable | String | The consumer key of the user. |
| batchJobId | Path Variable | String | The BatchJob identifier. |

Figure 23 – User Job Attributes

|  |  |  |
| --- | --- | --- |
| Name | Data Type | Description |
| id | String | The BatchJob identifier. |
| userId | String | The consumer key of the user. |
| businessApplicationName | String | The name of the business application. |
| businessApplicationVersion | String | The version of the business application. |
| <app job parameters> | *Varies* | Any business application specific job parameters will be returned. |
| jobStatus | String | The current status of the batch job. |
| millisecondsUntilNextCheck | Integer | The number of milliseconds the client should wait before requesting the status again. |
| numSubmittedRequests | Integer | The number of submitted requests. |
| numCompletedRequests | Integer | The number of successful completed requests. |
| numFailedRequests | Integer | The number of failed requests. |
| resultDataContentType | String | The content type requested of the result file. |
| resultsUrl | String | The URL to the user jobs results resource if the job has been completed. |

##### User Job Results

The user job results resource is a container for the result files for a user’s job. The representation returns links to the user job result resource for each of the result files.

|  |  |
| --- | --- |
| Path | /ws/users/{userId}/jobs/{batchJobId}/results |
| Method | GET |
| Content Types | text/html, text/xml, application/json, application/vnd.sun.wadl+xml, text/uri-list |
| Representation | Resource list |

Figure 24 – User Job Results Parameters

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Parameter Type | Data Type | Description |
| userId | Path Variable | String | The consumer key of the user. |
| batchJobId | Path Variable | String | The BatchJob identifier. |

##### User Job Result

The user job result resource represents a result file for a user’s job. The representation contains the resultData from a BatchJobResult.

|  |  |
| --- | --- |
| Path | /ws/users/{userId}/jobs/{batchJobId}/results/{resultId} |
| Method | GET |
| Content Types | *The* resultDataContentType *on the* BatchJobResult*.* |
| Representation | *The content of the* BatchJobResult *file.* |

Figure 25 – User Job Result Parameters

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Parameter Type | Data Type | Description |
| userId | Path Variable | String | The consumer key of the user. |
| batchJobId | Path Variable | String | The BatchJob identifier. |
| resultId | Path Variable | String | The BatchJobResult identifier. |

##### User Authenticated

The user authenticated resource checks to see if the user has been authenticated (logged in). If the user is logged in the authenticated object is returns. If the user is not authenticated they will be prompted to login.

This resource can be used by JavaScript applications to check that the user is logged in before submitting new job requests.

|  |  |
| --- | --- |
| Path | /ws/authenticated |
| Method | GET |
| Content Types | text/html, text/xml, application/json |
| Representation | Object Detail Authenticated |

Figure 26 – User Authenticated Attributes

|  |  |  |
| --- | --- | --- |
| Name | Data Type | Description |
| authenticated | boolean | True if the user is logged in, not returned if the user is not logged in |

#### Internal Web Services

The internal web services are used for communication between the components of the CPF. Access to these web services is limited to users with user\_account\_class <http://open.gov.bc.ca/cpf/SystemUser>.

##### User Attributes

The user attributes resource returns information about a user such as name, phone numbers and email addresses.

Each business application can be configured with a different security service. Therefore the user’s attributes can vary from application to application.

|  |  |
| --- | --- |
| Path | /ws/internal/apps/{businessApplicationName}/users/{userId}/attributes |
| Method | GET |
| Content Types | text/html, text/xml, application/json |
| Representation | Object Detail for UserAttributes |

Figure 27 – User Attributes Parameters

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Parameter Type | Data Type | Description |
| businessApplicationName | Path Variable | String | The name of the business application. |
| userId | Path Variable | String | The consumer key of the user. |

Figure 28 – WebADE User Attributes

|  |  |  |
| --- | --- | --- |
| Name | Data Type | Description |
| businessApplicationName | String | The name of the business application. |
| userId | String | The consumer key of the user. |
| sourceDirectory | String | The source directory the user account is from (e.g. IDIR, BCEID). |
| userTypeCode | String | The type of user (GOV or BUP). |
| userName | String | The user identifier (e.g. IDIR\JDOE). |
| userGuid | String | The GUID for the user. |
| firstName | String | The user’s first name. |
| middleInitial | String | The user’s middle initial. |
| lastName | String | The user’s last name. |
| displayName | String | The user’s full name. |
| emailAddress | String | The user’s email address. |
| phoneNumber | String | The user’s phone number. |
| expiryDate | String | The date the user expires. |

##### User Is Member of Group

The user is member of group resource returns a flag indicating if the user is a member of a group.

Each business application can be configured with a different security service. Therefore the user’s group membership can vary from application to application.

|  |  |
| --- | --- |
| Path | /ws/internal/apps/{businessApplicationName}/users/{userId}/groups/{groupName}/memberOf |
| Method | GET |
| Content Types | text/html, text/xml, application/json |
| Representation | Object Detail for GroupMembership |

Figure 29 – User is Member of Group Parameters

| Name | Parameter Type | Data Type | Description |
| --- | --- | --- | --- |
| businessApplicationName | Path Variable | String | The name of the business application. |
| userId | Path Variable | String | The consumer key of the user. |
| groupName | Path Variable | String | The name of the group. |

Figure 30 – GroupMembership Attributes

|  |  |  |
| --- | --- | --- |
| Name | Data Type | Description |
| businessApplicationName | String | The name of the business application. |
| userId | String | The consumer key of the user. |
| groupName | String | The name of the group. |
| memberOfGroup | Boolean | True if the user is a member of the group, false otherwise. |

##### User Can Perform Action

The user can perform action resource returns a flag indicating if the user can perform the named action.

Each business application can be configured with a different security service. Therefore the user’s action permissions can vary from application to application.

|  |  |
| --- | --- |
| Path | /ws/internal/apps/{businessApplicationName}/users/{userId}/actions/{actionName}/hasAccess |
| Method | GET |
| Content Types | text/html, text/xml, application/json |
| Representation | Object Detail for ActionPermission |

Figure 31 – User Can Perform Action Parameters

| Name | Parameter Type | Data Type | Description |
| --- | --- | --- | --- |
| businessApplicationName | Path Variable | String | The name of the business application. |
| userId | Path Variable | String | The consumer key of the user. |
| actionName | Path Variable | String | The name of the action. |

Figure 32 – ActionPermission Attributes

|  |  |  |
| --- | --- | --- |
| Name | Data Type | Description |
| businessApplicationName | String | The name of the business application. |
| userId | String | The consumer key of the user. |
| actionName | String | The name of the action. |
| hasAccess | Boolean | True if the user can perform the action, false otherwise. |

##### User Can Access Resource

The user can access resource resource returns a flag indicating if the user can access the resource.

Each business application can be configured with a different security service. Therefore the user’s resource permissions can vary from application to application.

|  |  |
| --- | --- |
| Path | /ws/internal/apps/{businessApplicationName}/users/{userId}/resources/{resourceClass}/ids/{resourceId}/actions/{actionName}/hasAccess |
| Method | GET |
| Content Types | text/html, text/xml, application/json |
| Representation | Object Detail for ResourcePermission |

Figure 33 – User Can Access Resource Parameters

| Name | Parameter Type | Data Type | Description |
| --- | --- | --- | --- |
| businessApplicationName | Path Variable | String | The name of the business application. |
| userId | Path Variable | String | The consumer key of the user. |
| resourceClass | Path Variable | String | The class of resource (e.g. table name or Java class name). |
| resourceId | Path Variable | String | The unique identifier of the resource (e.g. primary key value from table row). |
| actionName | Path Variable | String | The name of the action to perform on the resource, or any for permission for all actions. |

Figure 34 – ResourcePermission Attributes

|  |  |  |
| --- | --- | --- |
| Name | Data Type | Description |
| businessApplicationName | String | The name of the business application. |
| userId | String | The consumer key of the user. |
| resourceClass | String | The class of resource (e.g. table name or Java class name). |
| resourceId | String | The unique identifier of the resource (e.g. primary key value from table row). |
| actionName | String | The name of the action to perform on the resource, or any for permission for all actions. |
| hasAccess | Boolean | True if the user can access the resource, false otherwise. |

##### Worker Ping

The worker ping resource updates the worker’s last request time. CPF workers invoke this web service to indicate that they are still active even if they are not accepting any new work.

If the worker does not access the worker ping or next batch job execution group ID resource within a certain time period then the worker will be assumed to be dead and the work allocated to the worker will be distributed to other workers.

|  |  |
| --- | --- |
| Path | /ws/internal/workers/{workerId}/ping |
| Method | POST |
| Content Types | text/html, text/xml, application/json |
| Representation | Object Detail for WorkerPing |

Figure 35 – Worker Ping Parameters

| Name | Parameter Type | Data Type | Description |
| --- | --- | --- | --- |
| workerId | Path Variable | String | The unique identifier of the worker. |

Figure 36 – WorkerPing Attributes

|  |  |  |
| --- | --- | --- |
| Name | Data Type | Description |
| workerId | String | The unique identifier of the worker. |

##### Next Batch Job Execution Group ID

The next batch job execution group ID resource returns the next batch job execution group id to be executed for one of the specified business applications.

If the worker does not access the worker ping or next batch job execution group resource ID within a certain time period then the worker will be assumed to be dead and the work allocated to the worker will be distributed to other workers.

|  |  |
| --- | --- |
| Path | /ws/internal/workers/{workerId}/jobs/groups/nextId |
| Method | POST |
| Content Types | text/html, text/xml, application/json |
| Representation | Object Detail for ExecutionGroupId |

Figure 37 – Next Batch Job Execution Group ID Parameters

| Name | Parameter Type | Data Type | Description |
| --- | --- | --- | --- |
| workerId | Path Variable | String | The unique identifier of the worker. |
| businessApplicationName | Request Parameter | String | The name of the business applications to get execution groups for. Each business application name is specified as a separate request parameter. |

Figure 38 – ExecutionGroupId Attributes

|  |  |  |
| --- | --- | --- |
| Name | Data Type | Description |
| workerId | String | The unique identifier of the worker. |
| userId | String | The consumer key of the user. |
| businessApplicationName | String | The name of the business application. |
| batchJobId | Long | The unique identifier of the batch job. |
| groupId | String | The unique identifier of the batch job execution group. |
| groupUrl | String | The URL to get the details of the execution group. |

##### Batch Job Execution Group

The batch job execution group resource returns the details of the batch job execution group and the requests to be executed in the group.

|  |  |
| --- | --- |
| Path | /ws/internal/workers/{workerId}/jobs/{batchJobId}/groups/{groupId} |
| Method | GET |
| Content Types | text/html, text/xml, application/json |
| Representation | Object Detail for ExecutionGroup |

Figure 39 – Batch Job Execution Group Parameters

| Name | Parameter Type | Data Type | Description |
| --- | --- | --- | --- |
| workerId | Path Variable | String | The unique identifier of the worker. |
| batchJobId | Path Variable | String | The unique identifier of the batch job. |
| groupId | Path Variable | String | The unique identifier of the batch job execution group. |

Figure 40 – ExecutionGroup Attributes

|  |  |  |
| --- | --- | --- |
| Name | Data Type | Description |
| workerId | String | The unique identifier of the worker. |
| groupId | String | The unique identifier of the batch job execution group. |
| userId | String | The consumer key of the user. |
| batchJobId | Long | The unique identifier of the batch job. |
| businessApplicationName | String | The name of the business application. |
| applicationParameters | Map | Map of key value pairs for the business application specific job parameters. |
| requests | List<Map> | List of maps for the business application specific request parameters. Each entry in the list contains a map of parameters for one request to be executed by the business application. The map includes the requestId parameter containing the business application request’s unique identifier. |

##### Batch Job Execution Group Results

The batch job execution group results resource provides access to upload the results from the CPF worker after the execution of all the batch job requests in the group.

|  |  |
| --- | --- |
| Path | /ws/internal/workers/{workerId}/jobs/{batchJobId}/groups/{groupId}/results |
| Method | POST |
| Content Types | text/html, text/xml, application/json |
| Representation | Object Detail for ExecutionGroupResultsConfirmation |

Figure 41 – Batch Job Execution Group Results Parameters

| Name | Parameter Type | Data Type | Description |
| --- | --- | --- | --- |
| workerId | Path Variable | String | The unique identifier of the worker. |
| batchJobId | Path Variable | String | The unique identifier of the batch job. |
| groupId | Path Variable | String | The unique identifier of the batch job execution group. |
| results | Request Body | ExecutionGroupResults | The map containing the results. |

Figure 42 – ExecutionGroupResults Attributes

|  |  |  |
| --- | --- | --- |
| Name | Data Type | Description |
| batchJobId | Long | The unique identifier of the batch job. |
| groupId | String | The unique identifier of the batch job execution group. |
| results | List<Map> | List of results, contains a map with the values below for each reqyest |
| results[\*].requestId | Long | The unique identifier of the batch job request. |
| results[\*].perRequestResultData | boolean | Flag if the business application returned opaque output data. |
| results[\*].resultFields | Map | Map of result field values returned from executing the request by the business application. |
| results[\*].errorCode | String | The error code if there was an error executing the request. |
| results[\*].errorMessage | String | The error message if there was an error executing the request. |
| results[\*].errorTrace | String | The error stack trace if there was an error executing the request. |

Figure 43 – ExecutionGroupResultsConfirmation Attributes

|  |  |  |
| --- | --- | --- |
| Name | Data Type | Description |
| workerId | String | The unique identifier of the worker. |
| batchJobId | Long | The unique identifier of the batch job. |
| groupId | String | The unique identifier of the batch job execution group. |
| requestId | String | The unique identifier of the batch job request. |

##### Batch Job Request Opaque Input Data

The batch job request opaque input data resource provides access to the opaque input data of a batch job request. The representation is either a binary stream containing the inputData or a redirect to an inputDataUrl that contains the binary stream.

|  |  |
| --- | --- |
| Path | /ws/internal/workers/{workerId}/jobs/{batchJobId}/groups/{groupId}/requests/{requestId}/inputData |
| Method | GET |
| Content Types | *The* inputDataContentType *of the* BatchJobRequest*.* |
| Representation | *The* inputData *of the* BatchJobRequest or a HTTP 303 See Other redirect to the inputDataUrl. |

Figure 44 – Batch Job Request Opaque Output Data Parameters

| Name | Parameter Type | Data Type | Description |
| --- | --- | --- | --- |
| workerId | Path Variable | String | The unique identifier of the worker. |
| batchJobId | Path Variable | String | The unique identifier of the batch job. |
| groupId | Path Variable | String | The unique identifier of the batch job execution group. |
| requestId | Path Variable | String | The unique identifier of the batch job request. |

##### Batch Job Request Opaque Output Data

The batch job request opaque output data resource provides access to upload the opaque output data of a batch job request from the business application.

|  |  |
| --- | --- |
| Path | /ws/internal/workers/{workerId}/jobs/{batchJobId}/groups/{groupId}/requests/{requestId}/oututData |
| Method | POST |
| Content Types | text/html, text/xml, application/json |
| Representation | Object Detail for OpaqueOutputDataResults |

Figure 45 – Batch Job Request Opaque Input Data Parameters

| Name | Parameter Type | Data Type | Description |
| --- | --- | --- | --- |
| workerId | Path Variable | String | The unique identifier of the worker. |
| batchJobId | Path Variable | String | The unique identifier of the batch job. |
| groupId | Path Variable | String | The unique identifier of the batch job execution group. |
| requestId | Path Variable | String | The unique identifier of the batch job request. |

Figure 46 – OpaqueOutputDataResults Attributes

| Name | Data Type | Description |
| --- | --- | --- |
| workerId | String | The unique identifier of the worker. |
| batchJobId | Long | The unique identifier of the batch job. |
| groupId | String | The unique identifier of the batch job execution group. |
| requestId | String | The unique identifier of the batch job request. |

## Engineering Viewpoint

The engineering viewpoint describes how the components used in the system are distributed across the various servers in the Government environment.

The following diagram shows the deployment of the CPF components on the Government’s infrastructure. The CPF application contains the CPF App WAR and CPF Worker WAR components to be deployed to the Government application server. Additional CPF Worker WAR instances can be deployed to other application servers to handle increased load if required. In addition the CPF Worker WAR can be removed from the main application server if it causes too much load on that server.

The CPF plug-in are deployed to a shared directory. The CPF App WAR and CPF Worker WAR components load the plugins from this directory using a custom class-loader. This prevents the need to rebuild the CPF application for each new business application. A simple web application restart is all that is required.

The CPF App WAR uses the BMG series of database servers to store the data for the application.

The CPF Worker WAR does not have direct access to the database. Instead all communication is via the Internal Web Services on the CPF App WAR.

The CPF App WAR or CPF Worker WAR can be deployed to any OC4J (10.1.3) or Tomcat (6+) servlet container. They can also be deployed to the Amazon Cloud or a private Ubuntu Enterprise Cloud instance. Some business applications may not be deployable to the cloud as they require access to servers that are inside the private Government network.

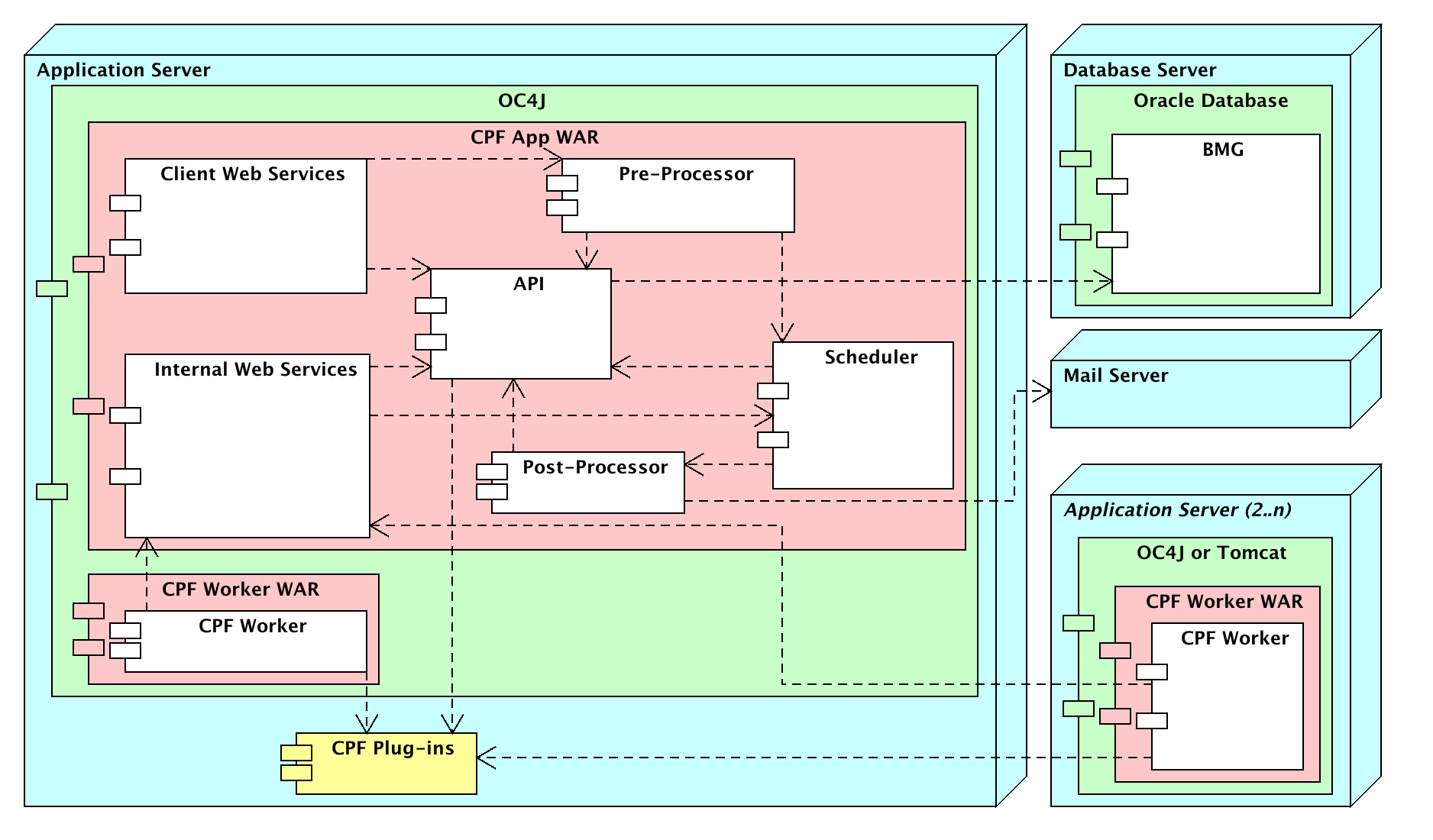


Figure 47 – Engineering Viewpoint

## Technology Viewpoint

### Component Technologies

The following table summarises the external technologies required for each of the components of the system. It does not include any software libraries that are deployed as part of the application package.

Table 7 – Component Technologies

|  |  |
| --- | --- |
| Server | Technology |
| Database Server (BMG) | * Oracle Database Server (>= 10.2.0.3) * Database storage for tables and indexes (>= 1GB) * RAM (>= 1GB) |
| Application Server | * OC4J (>= 10.1.3) * JDK (>= 1.5) * RAM (>= 2GB) |

### Interface Protocols

The following table summaries the protocols used by the interfaces exposed by the application

Table 8 – Component Technologies

|  |  |  |
| --- | --- | --- |
| Interface | Protocol | Security |
| Client Web Services | HTTP(s)/REST | Digest, OAuth 1.0, Siteminder |
| Internal Web Services | HTTP(s)/REST | Digest, OAuth 1.0 |

### Servers

The following table summarises the J2EE application servers used in each environment.

Table 9 – Servers

|  |  |  |  |
| --- | --- | --- | --- |
|  | Delivery | Test | Production |
| Database Server (BMG) | fry.geobc.gov.bc.ca | fry.geobc.gov.bc.ca | scorch.geobc.gov.bc.ca |
| Application Server | delivery.apps.gov.bc.ca | test.apps.gov.bc.ca | apps.gov.bc.ca |
| Email (SMTP) | apps.smtp.gov.bc.ca | apps.smtp.gov.bc.ca | apps.smtp.gov.bc.ca |

### Database Instances

The following table summarises the database instance used in each environment.

Table 10 – Database Instances

|  |  |  |  |
| --- | --- | --- | --- |
| Server Role | Delivery | Test | Production |
| BMG Operational | BMGDLV | BMGTST | BMGPRD |

### Network Shares

No network shares are required for the core CPF application.

# Data Design

## Data Security

The CPF database schema created and owned by the CPF user account. As this user is the owner of schema it has permission to manage all the CPF database objects. This account must only be used to create/manage the database tables and not by any applications.

The CPF application components use the PROXY\_CPF\_WEB user account with the CPF\_USER role to access the CPF schema. The permissions a web user has for the data is enforced at the application rather than the database level.

No other user accounts should require access to the CPF schema. Administrators and the developer may be granted the CPF\_VIEWER role to be able to view the data in the database for issue resolution purposes.

**Error! Reference source not found.** shows the database roles for the CPF application Table 13 shows the permissions the roles have for the CPF schema objects. The description of the permission flags is shown in the following table.

Table 11 – Database Roles

|  |  |
| --- | --- |
| Role | Description |
| CPF\_USER | Used by the application (proxy) user accounts that have permission to CRUD any tables. |
| CPF\_VIEWER | Used by the user accounts with read only access to the CPF schema for issue resolution. |

Table 12 – Database Permission Flags

|  |  |
| --- | --- |
| Flag | Description |
| M | Permission to manage the object. |
| C | Permission to create new data for the object. |
| R | Permission to read data from the object. |
| U | Permission to update existing data in the object. |
| D | Permission to delete data from the object. |

Table 13 – CPF Data Security Matrix

| Type | Object Name | CPF | CPF\_VIEWER | CPF\_USER |
| --- | --- | --- | --- | --- |
| Table | CPF.BATCH\_JOBS | M | R | CRUD |
| Table | CPF.BATCH\_JOB\_REQUESTS | M | R | CRUD |
| Table | CPF.BATCH\_JOB\_RESULTS | M | R | CRUD |
| Table | CPF.CONFG\_PROPERTIES | M | R | CRUD |
| Table | CPF.GRANTED\_PERMISSIONS | M | R | CRUD |
| Table | CPF.RESOURCE\_HISTORIES | M | R | CRUD |
| Table | CPF.SECURITY\_GROUPS | M | R | CRUD |
| Table | CPF.SECURITY\_GROUP\_MEMBERSHIPS | M | R | CRUD |
| Table | CPF.TIME\_PERIOD\_STATISTICS | M | R | CRUD |
| Table | CPF.USER\_ACCOUNTS | M | R | CRUD |

## Data Model Design

The data design for the CPF application is managed in the CPF container in the Ministry's Oracle designer repository. The physical data model has not been duplicated in this document instead the following physical data model documents should have been distributed with this design document or can be obtained from the designer repository web access.

* Server Model Diagram
* Table Definition Report
* Database Table and Index Size Estimates

## Warehouse Publication and Consumption

No warehouse publication or consumption is required by the CPF application.

# Human Interface Design

The CPF application follows the Government standards for web applications and pages. The application uses the standard template for HTML pages.

The body content of the web service pages (in the template) is automatically generated using the mapping described in sections 3.3.2.1.1.1 HTML Resource List and 3.3.2.1.2.1 HTML Object Detail. The descriptions of the resource contents are described in the interface definitions in section 3.3.2.2.

The following diagram shows the hierarchy of client web services resources (interfaces). Each node in the tree represents a path of one of the interfaces described in section 3.3.2.2.

Figure 48 – Client Web Services Resource Tree

# Technical Requirements

## Performance Requirements

The following performance requirements are for the core CPF application. Business application plug-ins will need to specify performance requirements in addition to those listed here.

|  |  |
| --- | --- |
| Speed | * Pages should take < 5 seconds in production to be completely displayed in a web browser[[5]](#footnote-5) over a LAN connection[[6]](#footnote-6). * The web server must support at least 100 concurrent requests in production (includes static items such as images as well as dynamic pages) and 50 concurrent transactions[[7]](#footnote-7) per second in production. * The CPF pre-processor, scheduler and post-processor must support a combined pool of 100 concurrent threads. * The CPF worker must support a pool of 100 concurrent threads for execution of business application requests. |
| Safety | * Application has no direct safety requirements affecting people, property or the environment. |
| Precision | * The application must ensure that all successfully submitted batch job requests are executed by a business application plug-in. |
| Reliability, Availability | * Best effort availability 24 x 7, support hours are 8:30-4:30 regular business days, no guarantees provided outside these hours. * Scheduled downtime following Ministry standard application and database maintenance schedules. * Response to emergency downtime will follow Ministry standard support schedules. * In the case of system failure, the application: * Must not retain data from incomplete actions (e.g. actions must be completed in database transactions). * Must be able to re-process failed pre-process, scheduled or post-process transactions upon restart. * No guarantees are made within the ministry's environment regarding the timeliness or success of delivering email via SMTP. |
| Capacity | * 10,000 batch jobs per business day with an average of 10 requests per order. * Product configuration data should be less than 10 MB. * Storage of batch job request and result data for 7 days. * Based on an average data size of 20 KB per batch job, average storage is 1.4 GB. * The metrics for the CPF workspace in the Oracle Designer Repository provide more detailed capacity requirements. |

## Quality Assurance

The following will be used to ensure the quality of the application and delivery.

* All code will be maintained using a Subversion repository. A Subversion tag will be created for all releases, so as to be able to create that release at a later date.
* All code will be reviewed to ensure each class and method has the appropriate Javadoc, comments and redundant code is removed.
* All code will be auto formatted using the Eclipse code formatter.

1. The actual physical data model description is contained within the reports generated from the Oracle designer repository that should be delivered with this document. [↑](#footnote-ref-1)
2. <http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html> [↑](#footnote-ref-2)
3. <http://www.json.org/> [↑](#footnote-ref-3)
4. <http://en.wikipedia.org/wiki/Web_Application_Description_Language> [↑](#footnote-ref-4)
5. If a page contains a lot of data (> 100 records) this time will be increased [↑](#footnote-ref-5)
6. Response times using third party dial-up or ADSL connections may be longer [↑](#footnote-ref-6)
7. A transaction is a dynamically generated HTML page or action (e.g. a form submission). [↑](#footnote-ref-7)