Detailed System Design

June 13, 2014

Version 1.12

Department of Health and Human Services
Administration for Children and Families
Office of Child Support Enforcement
<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Description of Change</th>
<th>Approval</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>12/18/2006</td>
<td>Initial Document</td>
<td>Rama K.</td>
</tr>
<tr>
<td>1.0</td>
<td>01/25/2007</td>
<td>Created Section 2, J2EE standards and updated documentation after team review</td>
<td>Chris M.</td>
</tr>
<tr>
<td>1.0</td>
<td>02/05/2007</td>
<td></td>
<td>Chris M.</td>
</tr>
<tr>
<td>1.1</td>
<td>02/08/2007</td>
<td></td>
<td>Chris M.</td>
</tr>
<tr>
<td>1.2</td>
<td>03/01/2007</td>
<td></td>
<td>Chris M.</td>
</tr>
<tr>
<td>1.3</td>
<td>03/29/2007</td>
<td>Format and Fonts, Removed embedded documents, created tables and added into Appendix A, B, C.</td>
<td>Duke T.</td>
</tr>
<tr>
<td>1.4</td>
<td>04/23/2007</td>
<td>Create new sections 4.1,2,3,4 Added Document Generation Cache Management, Exception Processing Framework</td>
<td>Chris M.</td>
</tr>
<tr>
<td>1.5</td>
<td>01/22/2008</td>
<td>Added Use Cases narratives to Section 4, Added text to Sections 2, 3 and 7</td>
<td>Ron D.</td>
</tr>
<tr>
<td>1.6-1.9</td>
<td>12/26/2013</td>
<td>New Format</td>
<td>TM</td>
</tr>
<tr>
<td>1.10 - 1.11</td>
<td>03/17/2014</td>
<td>Quality Assurance</td>
<td>CP</td>
</tr>
<tr>
<td>1.12</td>
<td>06/13/2014</td>
<td>Quality Assurance</td>
<td>CP</td>
</tr>
</tbody>
</table>
Table of Contents

1 Document Overview........................................................................................................... 7

2 Scope of Document .............................................................................................................. 7
2.1 Audience .......................................................................................................................... 7
2.2 Related Documentation .................................................................................................... 7
2.3 Definitions and Acronyms ............................................................................................... 8
2.4 Document Conventions .................................................................................................. 8

3 System Design .................................................................................................................... 9
3.1 Description ....................................................................................................................... 9
3.2 System Architecture ....................................................................................................... 12
3.2.1 Deployment Architecture View .................................................................................. 13

4 Software Design .................................................................................................................. 17
4.1 Design View .................................................................................................................... 19
4.1.1 Client-Side Data Representation .............................................................................. 19
4.2 Design Model ................................................................................................................ 20
4.2.1 Service to Worker .................................................................................................. 21
4.3 Controller ....................................................................................................................... 22
4.3.1 Managing Commands with a Controller ............................................................... 22
4.4 J2EE Application Tier ................................................................................................... 29
4.5 J2EE Design and Development .................................................................................... 29
4.5.1 Relational Persistence ............................................................................................. 29
4.5.2 Package Structure (Java Classes) .......................................................................... 30
4.5.3 Web Service Generation ......................................................................................... 32
4.6 Java Conventions and Standards ................................................................................... 34
4.6.1 Naming Conventions .............................................................................................. 34
4.6.2 Java Source Files ..................................................................................................... 50
4.6.3 Program Statements ................................................................................................. 53
4.6.4 Layout and Comments ............................................................................................ 66
4.7 Exception Handling ...................................................................................................... 85
4.7.1 Error Messages ......................................................................................................... 85
4.7.2 Exception Handling & Logging Standards ............................................................... 86
4.8 COTS Packages ............................................................................................................. 87
4.9 Components .................................................................................................................. 87
4.9.1 Component ID and Component Name .................................................................... 88
4.10 Interfaces ....................................................................................................................... 88

5 Detailed Design ................................................................................................................... 89
5.1 Case Management .......................................................................................................... 91
5.1.1 MTS Use Case: Case Consolidation ...................................................................... 92
5.1.2 MTS Use Case: Case Information Request ............................................................ 93
5.1.3 MTS Use Case: Case Intake .................................................................................... 94
5.1.4 MTS Use Case: Case Management ...................................................................... 95
5.1.5 MTS Use Case: Case Quick Locate ...................................................................... 96
Detailed System Design

5.1.6 MTS Use Case: Case Referral Process ................................................................. 97
5.1.7 MTS Use Case: Participant Demographics .......................................................... 98
5.1.8 MTS Use Case: Participant Income and Assets .................................................... 99
5.1.9 MTS Use Case: Participant Address ................................................................. 100
5.2 Locate Management ............................................................................................. 101
5.2.1 MTS Use Case: Locate .................................................................................... 102
5.3 Paternity ............................................................................................................... 103
5.3.1 MTS Use Case: Paternity Data Entry ................................................................. 104
5.3.2 MTS Use Case: Paternity Documents ............................................................... 105
5.3.3 MTS Use Case: Paternity Facilitation ............................................................... 106
5.3.4 MTS Use Case: Paternity Genetic Testing .......................................................... 107
5.3.5 MTS Use Case: Paternity Stipulation .................................................................. 108
5.4 Order Establishment ............................................................................................. 110
5.4.1 MTS Use Case: Order Data Entry ..................................................................... 111
5.4.2 MTS Use Case: Order Facilitation ................................................................... 112
5.4.3 MTS Use Case: Order Stipulation .................................................................... 113
5.4.4 MTS Use Case: Order Documents ................................................................... 114
5.4.5 MTS Use Case: Order Monitor ........................................................................ 116
5.5 Enforcement ........................................................................................................ 117
5.5.1 MTS Use Case: Enforcement .......................................................................... 118
5.5.2 MTS Use Case: Enforcement Monitor .............................................................. 119
5.6 Financial Management ......................................................................................... 120
5.6.1 MTS Use Case: Financial Payment Posting ...................................................... 122
5.6.2 MTS Use Case: Account Charging .................................................................... 123
5.6.3 MTS Use Case: Financial Delinquency Monitor ................................................ 124
5.6.4 MTS Use Case: Monthly NCP Billing Notices ................................................... 125
5.6.5 MTS Use Case: Financial Account .................................................................... 126
5.6.6 MTS Use Case: Financial Account Summary ................................................... 127
5.6.7 MTS Use Case: Collection Posting ................................................................... 128
5.6.8 MTS Use Case: Deposit Processing .................................................................... 129
5.6.9 MTS Use Case: Financial Pass-through ............................................................ 130
5.6.10 MTS Use Case: Financial Distribution ............................................................ 131
5.6.11 MTS Use Case: Financial Disbursements ........................................................ 133
5.6.12 MTS Use Case: Bank File Extraction & Recon Matching .................................. 134
5.6.13 MTS Use Case: Bank Reconciliation Statement ............................................... 136
5.6.14 MTS Use Case: Monthly Check Escheatment ................................................. 138
5.6.15 MTS Use Case: Transaction Adjustments ....................................................... 139
5.6.16 MTS Use Case: Monthly Account History ....................................................... 140
5.6.17 MTS Use Case: Quarterly Payment History Notices ....................................... 141
5.7 Application Management .................................................................................... 143
5.7.1 MTS Use Case: Application jBPM Framework ................................................... 144
5.7.2 MTS Use Case: Application Cache .................................................................... 145
5.7.3 MTS Use Case: Application Data Validation ...................................................... 146
5.7.4 MTS Use Case: Application Exception Processing ............................................. 147
5.7.5 MTS Use Case: Application Scheduled Process ................................................. 148
5.7.6 MTS Use Case: UI Validators ............................................................................ 150
# Detailed System Design

5.7.7 MTS Use Case: Reporting ................................................................. 151
5.8 Document Generation ........................................................................... 152
5.8.1 MTS Use Case: Document Data Merge ........................................... 154
5.8.2 MTS Use Case: Document Mapping .................................................... 154
5.8.3 MTS Use Case: Document Print Management .................................... 155
5.8.4 MTS Use Case: Document Generation – Admin Templates ............... 155
5.8.5 MTS Use Case: Disbursement Check Printing ..................................... 156
5.9 Security and Administration ................................................................. 157
5.9.1 MTS Use Case: Security Administration ........................................... 158
5.9.2 MTS Use Case: User Access ............................................................... 159
5.9.3 MTS Use Case: User Calendar ............................................................ 160
5.9.4 MTS Use Case: User Tickler & Worklists ........................................... 161
5.9.5 MTS Use Case: System Configuration ............................................... 163
5.9.6 MTS Use Case: Security Privileges .................................................... 165
5.10 Common Functions ............................................................................. 166
5.10.1 MTS Use Case: Common Monitor .................................................... 167
5.10.2 MTS Use Case: Member Match .......................................................... 175
5.10.3 MTS Use Case: Create Financial Accounts ....................................... 177
5.10.4 MTS Use Case: Create Debt Accounts .............................................. 178
5.11 Third Party Data Management ............................................................. 179
5.11.1 MTS Use Case: Third Party Maintenance & Configuration ............... 181
5.11.2 MTS Use Case: Third Party Search .................................................... 182
5.11.3 MTS Use Case: Third Party Agency .................................................... 183
5.11.4 MTS Use Case: Third Party Attorney .................................................. 184
5.11.5 MTS Use Case: Third Party Correctional Facility ............................... 185
5.11.6 MTS Use Case: Third Party Court ....................................................... 186
5.11.7 MTS Use Case: Third Party Employer ............................................... 187
5.11.8 MTS Use Case: Third Party Financial Institution ............................... 188
5.11.9 MTS Use Case: Third Party Genetic Testing Lab ............................... 189
5.11.10 MTS Use Case: Third Party Health Care Provider ............................ 190
5.11.11 MTS Use Case: Third Party Health Insurance Provider ..................... 191
5.11.12 MTS Use Case: Third Party Law Enforcement ................................. 191
5.11.13 MTS Use Case: Third Party Probation Parole Officer ........................ 192
5.11.14 MTS Use Case: Third Party Registered Agent .................................. 193
5.11.15 MTS Use Case: Third Party School ................................................... 193
5.11.16 MTS Use Case: Third Party TCSE Office .......................................... 194

6 Data Design .......................................................................................... 195
6.1 Database Standards ............................................................................... 195
6.2 Database Design Based on Business Functions ...................................... 195
6.3 Database Table Layouts and Relationships .......................................... 195

7 System Performance ............................................................................... 196
7.1 Operating System Clustering ................................................................. 196
7.2 Application Server Clustering ............................................................... 197
7.3 Database Server Clustering ................................................................. 197
1 Document Overview
This document is the technical blueprint developed by the Office of Child Support Enforcement (OCSE), Division of State and Tribal Systems technical team for the Model Tribal System (MTS). The scope of this document is limited to design elements for MTS. The foundation for the current design documentation is based on the MTS Software Requirements Specification (SRS) and General System Design (GSD) and is intended to satisfy all the customer requirements, objectives, and expectations.

Systems design and development standards are defined and will be used during the implementation phase. Coding and syntax requirements are outlined with examples to promote code uniformity throughout the application.

Specific use case instances contain the low level design for all business processes defined in the SRS. Each use case is an embedded document with specific references to supporting design documentation for the specific design process.

2 Scope of Document
Please refer to the separate table, MTS IT Bill of Materials, in Appendix C.

2.1 Audience
This document is intended for the following audience:

- Designers and Developers.
- Technical Management Team.
- Business Team.

This document assumes that the reader is familiar with MTS requirements.

The skills required to understand this document are a familiarity with the following techniques, standards, and technologies:

- JBoss Application Server 4.0.5.
- jBPM.
- Adobe Flex 2.
- MySQL Server and Administration.
- Apache Web Server.
- Java 2 Enterprise Edition (J2EE) 1.4.
- Apache Web Services.

2.2 Related Documentation
Other system documentation for this system should include:
<table>
<thead>
<tr>
<th>Item</th>
<th>Title</th>
<th>Author</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Cases</td>
<td>MTS_UC_&lt;Use Case Name&gt;</td>
<td>MTS Project Team</td>
<td>Embedded within existing documentation</td>
</tr>
<tr>
<td>Software Architecture Document</td>
<td>MTS Architecture Design</td>
<td>MTS Project Team</td>
<td>Embedded within existing documentation</td>
</tr>
<tr>
<td>J2EE design Checklist</td>
<td>MTS_DES_checklist.doc</td>
<td>MTS Project Team</td>
<td></td>
</tr>
<tr>
<td>Reports Design Checklist</td>
<td>MTS_SystemDesignChecklist_Reports.doc</td>
<td>MTS Project Team</td>
<td></td>
</tr>
<tr>
<td>Report Design Standards</td>
<td>MTS_Reports_Design_Standards.doc</td>
<td>MTS Project Team</td>
<td></td>
</tr>
</tbody>
</table>

2.3 Definitions and Acronyms
Refer to Appendix A, Glossary and Acronyms.

2.4 Document Conventions

<table>
<thead>
<tr>
<th>Item</th>
<th>Format Convention</th>
<th>Sample</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keys pressed on the Keyboard</td>
<td>Enclosed in &lt; &gt;</td>
<td>&lt;Enter&gt; or &lt;Ctrl+F&gt;</td>
</tr>
<tr>
<td>Variable or text that is replaced with the other text</td>
<td>Enclosed in square brackets</td>
<td>[FIELD_NAME]</td>
</tr>
</tbody>
</table>
3 System Design

3.1 Description

The MTS application provides an integrated environment for tribal child support organizations responsible for managing child support cases. MTS is a new application with business functionality that will allow for efficient management of tribal child support enforcement cases. The system is being developed as a model for tribes to use as a baseline system that can be customized based on specific tribal requirements. Each instance of the system software is intended to support only one tribal organization.

The system performs the following high-level functions:

- It contains all data needed to manage a case, including records of payments and collections made by tribal financial organizations.
- It accepts applications for child support services and sets up child support cases.
- It locates participants in the case.
- It supports the establishment of paternity and the establishment of support orders.
- It supports the enforcement of child support orders.
- It supports the collection of payments, distribution, and disbursement of payments.
- The system supports the production of all required federal and tribal reports.
- The system ensures the privacy of all data.

The following are business processes that comprise functional areas of the MTS system design:

**Case Intake and Update**

Case Initiation and Update is the process of responding to a request for services. This includes opening and/or reopening a case and assessing and determining the next appropriate action on a case. It also includes entering data on a case, either at intake or at any later time in the life of the case.

**Locate**

Locate is the process of gathering information concerning the physical location and verified mailing address of a Custodial Parent (CP) or Noncustodial Parent (NCP). It also includes information on the parties’ employer(s), income, or assets. Locate information is used to support the efficient collection of child support.

**Paternity Establishment**

Paternity Establishment is the process of obtaining a voluntary acknowledgement of or a stipulation to paternity. It also includes the legal processes needed to obtain a finding of paternity by a court.
Order Establishment
Order Establishment is the process of obtaining a voluntary stipulation to child support by an NCP or conducting the legal processes needed to obtain a child support order from a court. It also includes the Review and Adjustment process.

Enforcement
Enforcement is any action that may be taken to ensure collection of a child support obligation. Enforcement includes income withholding, license suspension, liens on property, and various other tribal, state and federal remedies.

Case Management
Case Management includes numerous actions affecting the status or organization of an established case. It includes redirecting payments, consolidating a case, transferring a case, and closing a case. It also includes monitoring responses to actions on a case and determining the next action that should be taken on a case.

Financial Management
Financial Management includes all the processes involve in collecting and paying out child support monies. The system must conduct these functions in accordance with accepted accounting practices. The functions include:

Obligation Management
Obligation Management is the process of managing owed amounts, primarily court ordered support (financial and non-financial) owed by the NCP, but also including any judgments, fees, and other costs related to a child support case.

Collections
Collections is the process of receiving funds from the NCP and other payors, and all functions related to recording and depositing monies paid by NCPs for child support obligations.

Distribution
Distribution is the process of applying allocated collections to various debts and obligations. This capability also determines how collections will be applied to the obligor’s child support obligations.

Disbursement
Disbursement is the process of handling outbound payments to the appropriate parties.

System Security and Administration
System Security and Administration provide for maintaining the security and privacy of personal and financial data.
Security
The Security function will provide for restricting access to the MTS application and data to only authorized users under controlled circumstances. It will provide for separation of functions to ensure proper handling of financial accounts.

Administration
The Administration process provides for configuring the system to support a variety of tribal child support enforcement (TCSE) programs, each with different needs and different governing regulations. The system software will be extensively configurable at and subsequent to installation. Configuring and reconfiguring the system should have a minimum impact on program operations.

Reporting
Reporting includes extracting and abstracting system data to produce reports that ensure the reliability of the system’s financial processes and the business effectiveness of the child support program as a whole.
3.2 System Architecture

The MTS System Architecture document describes the operational structure of the various components that comprise various MTS environments. It focuses on the technological components of each environment and specifically describes the model system solution and information system architecture. It describes the different layers in the architecture for the MTS application (presentation tier, business tier, persistence tier and data tier). Each layer has a façade/controller object.

This subsection presents a high-level view of the system solution and information system architectures as well as the various deployment view architectures associated with:

- Development Configuration.
- UAT/Pre-Production Configuration.
- Production Configuration.

The information system architecture for MTS is depicted below:

![Figure 2.1 – MTS Information System Architecture](image-url)
The high-level system solution architecture for MTS is depicted below:

![MTS System Solution Architecture](image)

**Figure 2.2 – MTS System Solution Architecture**

The System Architecture document describes the architectural representation of the pilot system including network architecture, logical and physical system architecture, use cases to define the functional aspects, data relationships, and deployment view.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MTS_SystemArchitecture</td>
<td>Alfresco System -&gt; MTS -&gt; Design – Detail Design</td>
</tr>
</tbody>
</table>

### 3.2.1 Deployment Architecture View

The following subsection shows the deployment view of MTS for the various project-specific deployments. These views show how the various executables and components are mapped to the underlying OS and hardware platforms or computing nodes. It also depicts system hardware and software deployment and installation. The following deployments are depicted:

- Development Configuration.
- UAT/Pre-Production Configuration.
- Production Configuration.
The following are production configurations with variations on a clustered deployment. A clustering scheme provides high availability, fault tolerance, and increased scalability through the use of application servers. Workload management and request distribution is achieved through load balancers which provide scalability. Each server component can service any request from any management agent in any order.

The architectures below depict single server and multiple server vertical cluster environments (multiple application server instances hosted on the same physical machine [node]) and horizontal clusters (multiple physical machines [nodes]).
Figure 2.5 – Vertical Cluster Configuration (Single Server)

Figure 2.6 – Vertical Cluster Configuration (Multiple Servers)
The MTS application infrastructure tiers supported, including the Web Servers and the Application Servers, will be highly available. The MySQL database can optionally be deployed on a dedicated server.
4 Software Design

The software design is based on the Cairngorm framework which provides a Model View Controller (MVC) model for Rich Internet Application (RIA) development. The combination of Adobe Flex 2 and Flash Player provides a rich User Interface (UI) presentation and allows for the creation of a scalable application using the J2EE architecture for the application tier.

User presentation is displayed using the Adobe Flash Player within a standard browser. Flex 2 is an application framework that consists of Multimedia XML (MXML), Action Scripts (an ECMAScript-based scripting language), and a class library of UI components to aid design and development. The Cairngorm framework is based on the MVC design architecture. This architecture isolates data access and business logic from the UI and data presentation interactions. This is accomplished by the use of a common controller that performs actions and responds to events instantiated by the user that can make changes to the model. The combination of Flex 2 and Cairngorm provides the necessary tools and framework to create a highly scalable and flexible application. Figure 3.1 provides a diagram of all layers of the MTS application architecture.

Flex 2 user interface design is implemented based on the Cairngorm framework. The Cairngorm Microarchitecture is a lightweight yet prescriptive framework for RIA development.

Cairngorm Microarchitecture addresses the key areas in user interface design:

- Handling user interaction.
- Encapsulating business logic and server interactions.
- Managing state on the client.

The following design patterns are implemented in Cairngorm:

- Front Controller and Command patterns implement “Service to Worker” microarchitecture for listening and responding to user requests.
- Business Delegate and Service Locator patterns interact so that the business logic can be reused and also encapsulate it so that it establishes a clear contract between client and server-side development teams.
- Value Object pattern from J2EE collaborates with the Model Locator pattern in Cairngorm.
Cairngorm 2.0 Microarchitecture – Basic Server RPC

From ‘Developing Flex RIAs with Cairngorm Microarchitecture’ by Steven Webster

The EventBroadcaster dispatches the event to the front controller

Event and Data Payload are passed together to Front Controller

The Front Controller enlists the appropriate command

Business Delegate relays response back to command

Response is sent back to Business Delegate

Command updates Model

Business Delegate enlists the application-tier service and passes references of the Command’s onResult and onFault handlers

FDS/AMF, SOAP, XML-RPC, AMFPHP or XML:80

Updated: 11/23/2006 - Diagram by Evan Gifford (evan@usa.com) with special thanks to Steven Webster, Alex Uhlmann, Russell Munro, Jesse Warden, Dan Nielsen, Tom Chiverton and everyone on the flexcoders group – flexcoders.org

Figure 3.1 – Cairngorm Microarchitecture
4.1 Design View

The component-based model eases the creation of Flex 2 applications. Pre-built components included with Flex 2 can be used for user interface design, can extend components to add new properties and methods, and can create new components. To build an application using Flex 2, the user interface is defined using two types of visual components: controls and containers.

Controls are user interface components that handle user interactions and display data that users can manipulate directly through that control. Examples of controls are the DataGrid and the TreeControl.

A container defines a region of the Flash Player drawing surface and controls the layout for everything in the container, including other containers and controls. Examples of containers are a data entry Form, a Box, and a Grid.

Flex components are extremely flexible and provide developers with a great deal of control over the component’s appearance, the component’s reaction to user interactions, the font and font size of any text included in the component, and the size of the component in the application. Flex 2 components support the following characteristics:

- **Events** – Application or user actions that require a component response.
- **Behaviors** – Visible or audible changes to the component triggered by an application or user action.
- **Skins** – Symbols that control a component’s appearance.
- **Styles** – Set of characteristics, such as font, font size, and text alignment.
- **Size** – Height and width of a component (all components have a default size).

The characteristics of the controls can be controlled at development time through MXML or Cascading Style Sheets (CSS), or at runtime through the component’s ActionScript API, including creating or destroying instances of a component based on application data or user interaction.

4.1.1 Client-Side Data Representation

Client side data is represented using Value Objects. The value objects contain attributes and are also used to exchange or transfer data between different tiers within the application.

Whenever data is retrieved on the server-side business logic and transformed into a collection of value objects representing, for example, a list of cases for a participant or participants within a case, the same collection can be passed as value objects between the J2EE tier and the UI tier.

These business objects that represent the data must be named as `<valueobject>VO`, where valueobject is the data that represents the user’s view. Example: CaseVO is a data structure that represents the Case Detail or Case screen.
All Value Objects in MTS application must extend the BaseVO (Base Value Object) and implement the ValueObject, Comparable.

Example:

```java
class org.dhhs.mtcse.ui.vo.CaseVO extends BaseVO implements 
ValueObject, Comparable {
    public static var registered:Boolean =
        Object.registerClass( "org.dhhs.mtcse.beans.CaseVOType", CaseVO);
    public var long:caseID;
    public var status:String;
    public var type:String;
    ...
    ...
    ...
}
```

To allow Flex to map between client-side and server-side value objects, you must provide some hints to the Flex server about the relationship between the classes. In Cairngorm, you achieve this by advocating that each and every instance of a value object class contains the following:

```xml
<>
```

This line ensures that a CaseVO class on the client maps to the Java class CaseVO.java that resides in the package `org.dhhs.mtcse.beans.CaseVOType` on the server.

4.2 Design Model

Data binding allows two objects to act as a source and a destination. Changes to the source object are reflected immediately in the destination object.

The Model Locator pattern strategy encourages the use of data binding so that view components bind directly to the client-side state held in the single instance of the ModelLocator class. In this way, whenever the model is updated in ModelLocator, all view components binding to the model receive notifications (through the underlying data-binding mechanism) and update themselves to render the new model on the client.

Cairngorm provides a marker interface for the Model Locator pattern, and in the MTS UI we have a single class that implements this marker interface, `org.dhhs.mtcse.ui.model.ModelLocator`.

Having all the attributes on the Model Locator pattern as static attributes ensures that the Model Locator pattern is a simple implementation of a singleton.

Any ValueObject that you want to bind to a data control must be defined as an attribute in the ModelLocator if changes to the model are to be reflected in the control.
4.2.1 Service to Worker

Consider the events in the MTS application for createCas” and closeCase etc. The features map to events such as createCase, closeCase and so forth. Whenever a user action indicates the desire to execute a feature, Cairngorm requires that you broadcast an appropriate event.

In response to each event a feature is executed and these are called commands. Each and every command exposes a single entry point, a method called execute(), which allows a third party to invoke the command without understanding what the command accomplishes. Often these commands are called worker classes because they are responsible for actually carrying out the work behind the application.

MTS Login Command

```java
package org.dhhs.mtcse.ui.commands
{

    import mx.rpc.events.ResultEvent;
    import com.adobe.cairngorm.business.Responder;
    import com.adobe.cairngorm.commands.Command;
    import mx.controls.Alert;
    import mx.rpc.events.ResultEvent;
    import com.adobe.cairngorm.control.CairngormEvent;
    import org.dhhs.mtcse.ui.business.LoginDelegate;
    import org.dhhs.mtcse.ui.control.LoginEvent;
    import org.dhhs.mtcse.ui.model.ModelLocator;
    import org.dhhs.mtcse.ui.vo.UserVO;

    public class LoginCommand implements Command, Responder
    {

        private var model : ModelLocator = ModelLocator.getInstance();

        public function execute( event : CairngormEvent ) : void
        {
            model.login.isPending = true;

            var delegate : LoginDelegate = new LoginDelegate( this );
            var loginEvent : LoginEvent = LoginEvent( event );
            delegate.login( loginEvent.userVO);
        }
    }
}
```
public function onResult( event : * = null ) : void
{
    model.login.loginDate = new Date();
    model.login.userVO = UserVO( event );
    model.login.isPending = false;

    model.workflowState =
        ModelLocator.VIEWING_LOGGED_IN_SCREEN;
}

public function onFault( event : * = null ) : void
{
    model.login.statusMessage =
        "Your username or password was wrong, please try
        again."
    model.login.isPending = false;
    Alert.show( "Invalid Login!" );
}

In Cairngorm, this interface simply enforces the requirement that the command have a
method called execute() that can act as its entry point. This allows Cairngorm to
execute each and every command without caring about what the command actually does.

4.3 Controller

The Front Controller architectural design pattern centralizes an application's request
processing and view selection in a single component. The Front Controller receives
requests from the client (View) and dispatches them to the application model. This single
point of dispatch makes the Front Controller a logical place for such global facilities as
security and logging. The Front Controller also selects and formats the next client view.
The controller is also an application of the Mediator pattern, because it decouples view
components from one another.

4.3.1 Managing Commands with a Controller

Cairngorm introduces the Front Controller pattern as a single point of entry for all
Cairngorm events.

All event constants must be defined in the EventConstants action script file.

package org.dhhs.mtcse.ui.commands
/**
 * String constants for all application events.
 * @author Rama Kodumagulla
 *
 */

public class EventConstants
{

    /*
     * Constants for Cairngorm Events to dispatch commands
     */

    public static var EVENT_SEARCH_ROLE:String = "searchRoleEvent";
    public static var EVENT_SAVE_ROLE:String = "saveRoleEvent";
    public static var EVENT_ADD_ROLE:String = "addRoleEvent";
    public static var EVENT_DELETE_ROLE:String = "deleteRoleEvent";
    public static var EVENT_SEARCH_USER:String = "searchUserEvent";
    public static var EVENT_SAVE_USER: String = "saveUserEvent";
    public static var EVENT_ADD_USER: String = "addUserEvent";
    public static var EVENT_DELETE_USER: String = "deleteUserEvent";
    public static const EVENT_GETMENUS : String = "getMenusEvent";
    public static const CREATE_MENU : String = "createMenuEvent";
    public static const SAVE_MENU : String = "saveMenuEvent";
    public static const DELETE_MENU : String = "deleteMenuEvent";

}

This is a best-practice approach of naming all events as constants on the FrontController instance.

The code below defines the “Front Controller” pattern in the MTS application. The constructor ensures that when the application does create the controller, it creates it with the full knowledge of which commands it can delegate work to, depending on which events it hears being broadcast.

package org.dhhs.mtcse.ui.control
{
    import com.adobe.cairngorm.control.FrontController;
import org.dhhs.mtcse.ui.commands.*;
import mx.events.MenuEvent;
import org.dhhs.mtcse.ui.constants.ApplicationConstants;

public class Controller extends FrontController
{
    public function Controller()
    {
        // Login
        addCommand(LoginEvent.EVENT_LOGIN, LoginCommand );

        // Menu Maintenance
        addCommand(EventConstants.EVENT_GETMENUS, MenuCommand );
        addCommand(EventConstants.SAVE_MENU, MenuCommand);
        addCommand(EventConstants.CREATE_MENU, MenuCommand);
        addCommand(EventConstants.DELETE_MENU, MenuCommand);

        // Role Maintenance
        addCommand(SearchRoleEvent.SEARCH_ROLE, SearchRoleCommand );
        addCommand(SaveRoleEvent.SAVE_ROLE, SaveRoleCommand );
        addCommand(AddRoleEvent.ADD_ROLE, SaveRoleCommand );
        addCommand(DeleteRoleEvent.DELETE_ROLE, SaveRoleCommand );

        // User Maintenance
        addCommand(SearchUserEvent.SEARCH_USER, SearchUserCommand );
        addCommand(SaveUserEvent.SAVE_USER, SaveUserCommand );
        addCommand(AddUserEvent.ADD_USER, SaveUserCommand );
        addCommand(DeleteUserEvent.DELETE_USER, SaveUserCommand );
    }
}

By extending the base FrontController class in the Cairngorm framework, the addCommand() method of the FrontController class registers events with corresponding command classes. The underlying Cairngorm architecture does the rest of the work. It simply broadcasts the appropriate event from anywhere in your application and Cairngorm ensures that the relevant command is invoked.
Broadcasting Events

Cairngorm provides a singleton EventBroadcaster class that collaborates with the FrontController class. To dispatch Cairngorm events, dispatch them with the Cairngorm Event Broadcaster.

The following example dispatches the event from the Login panel to the Command pattern:

```ActionScript
public function loginUser() : void
{
    var userVO : UserVO = new UserVO();
    userVO.userID = userID.text;
    userVO.password = password.text;

    var event : LoginEvent = new LoginEvent( userVO );
    CairngormEventDispatcher.getInstance().dispatchEvent( event );
}
```

Integrating with J2EE

These are now known as Flex Remote Procedure Call (RPC) services:

- **HTTPService** – Passes textual data over HTTP. This is often used to fetch RSS feeds, read XML generated by a server, or simply pass text-based data from server to client.

- **WebService** – Allows a Flex application to invoke server-side methods using SOAP web services and handle the results of web service calls. These web services may reside on the server that has served the RIA or may call third-party web services through a proxy on the Flex server.

- **RemoteObject** – Allows a Flex application to directly invoke methods on Java classes residing on the application server that has served the RIA. Data transfer is in a binary format called Action Message Format (AMF) over HTTP or HTTPS, with the Flex server translating Java and ActionScript objects as they pass over the wire.

*Note: In MTS, only HttpService and WebService are being used.*

The Service Locator is a repository of all services that the RIA requires in which all services are centralized using service tags.

Furthermore Service Locator abstracts developers from whether a service is a Java object, a web service, or an HTTP service, and allows them to "treat them all the same." In essence, the Service Locator encapsulates the implementation details of services, so developers can simply request a service from the Service Locator, invoke methods on the service, and handle the results however they might be shipped over the wire.

Cairngorm provides a base-class, ServiceLocator, which is a singleton for service access. In a Cairngorm application, this base-class is extended through MXML, with the
convention of always having a file called Services.mxml in the business/package of the application.

Here is an example of Services.mxml for MTS application.

```xml
<?xml version="1.0" encoding="utf-8"?>
<cairngorm:ServiceLocator
    xmlns:mx="http://www.adobe.com/2006/mxml"
    xmlns:cairngorm="com.adobe.cairngorm.business.*">
    <mx:WebService id="menuManagerService"
        wsdl="http://localhost:8080/MTCSE_J2EEApp/services/MenuManagerWebService?wsdl" showBusyCursor="true">
        <mx:operation name="getMenus" resultFormat="e4x">
            ...
        </mx:operation>
        <mx:operation name="saveMenu" resultFormat="e4x">
            ...
        </mx:operation>
        <mx:operation name="createMenu" resultFormat="e4x">
            ...
        </mx:operation>
    </mx:WebService>

    <mx:WebService id="authenticationService"
        wsdl="http://localhost:8080/MTCSE_J2EEApp/services/AuthenticationServiceWebService?wsdl" showBusyCursor="true">
        <mx:operation name="authenticateUser" resultFormat="xml">
            ...
        </mx:operation>
    </mx:WebService>

    <mx:WebService id="userAdministrationService"
        <mx:operation name="getRoleByRoleID" resultFormat="e4x">
            ...
        </mx:operation>
        <mx:operation name="getUsersByFilter" resultFormat="e4x">
            ...
        </mx:operation>
        <mx:operation name="getUsers" resultFormat="e4x">
            ...
        </mx:operation>
        <mx:operation name="addUser" resultFormat="e4x">
            ...
        </mx:operation>
        <mx:operation name="saveRole" resultFormat="e4x">
            ...
        </mx:operation>
    </mx:WebService>
</cairngorm:ServiceLocator>
```
The Services.xml file always specifies result and fault handlers. The result handler is the method called when a service call returns some data. The fault handler is called when a service call fails. Because the Service Locator anticipates that its services will be used multiple times in a multitude of different ways, it makes no assumptions about how to handle any results that are returned. Instead, the Service Locator assumes that the result handler and the fault handler will belong to the class that called the service. It achieves this by specifying `event.call.resultHandler` and `event.call.faultHandler` as the handlers through a Flex object known as the pending call.

**Invoking a Service on the Service Locator**

The following method in the LoginCommand shows the service invocation:

```actionscript
public function execute( event : CairngormEvent ) : void
{
    model.login.isPending = true;

    var delegate : LoginDelegate = new LoginDelegate( this );
    var loginEvent : LoginEvent = LoginEvent( event );
    delegate.login( loginEvent.userVO );
}
```

**The Login Delegate sample**

```actionscript
package org.dhhs.mtcse.ui.business
{
    import com.adobe.cairngorm.business.Responder;
    import com.adobe.cairngorm.business.ServiceLocator;
    import org.dhhs.mtcse.ui.vo.UserVO;

    import flash.utils.setTimeout;
    import mx.rpc.events.FaultEvent;
    import mx.rpc.events.ResultEvent;

    import mx.rpc.AsyncToken;
    import mx.rpc.soap.mxml.WebService;
    import mx.rpc.AbstractOperation;
    import mx.controls.Alert;
    import org.dhhs.mtcse.ui.factories.MTCSEFactory;
    import mx.messaging.messages.SOAPMessage;

    public class LoginDelegate
    {
    }
}
```
private var responder : Responder;
private var service : WebService;

public function LoginDelegate( responder : Responder )
{
    this.service = ServiceLocator.getInstance().getService("authenticationService") as WebService;
    this.responder = responder;
}

public function login( userVO : UserVO ): void
{
    service.addEventListener(ResultEvent.RESULT, login_onResult, false, 0.0, false);
    service.addEventListener(FaultEvent.FAULT, login_onFault, false, 0.0, false);
    service.authenticateUser(userVO.userID, userVO.password);
}

protected function login_onResult(event:ResultEvent):void
{
    var xmlDoc:XML = new XML(event.result);

    var userVO:UserVO = MTCSEFactory.getInstance().buildUserVOFromXML(xmlDoc);
    responder.onResult(userVO);
}

protected function login_onFault(event:FaultEvent):void
{
    Alert.show(event.fault.toString());
    responder.onFault(new FaultEvent(FaultEvent.FAULT, false, true, event.fault));
}
4.4 J2EE Application Tier

The MTS will use a J2EE application development environment for the application tier. The Java Enterprise services will support the scalability requirements defined for the model system and allow Java programmers the flexibility to create programs based on existing design patterns and current frameworks.

4.5 J2EE Design and Development

Standardization of the J2EE environment can be established in many ways. MTS will utilize coding standards and toolsets to provide a scalable, well designed, structured environment. Java persistence will be maintained using Hibernate to generate a standard entity model. Java package structures will conform to defined naming structures and will contain related classes. All web services will be generated within the Eclipse Integrated Development Environment (IDE) for the associated application server.

J2EE standards will be founded on the use of standard design patterns using approaches that both simplify and standardize application building. The overall goal of pattern usage is to standardize the application's architectural layer and simplify the complexities of the underlying independent business processes. Standardization of the programming within these Java programs will further enhance system standardization and simplify modification/maintenance.

4.5.1 Relational Persistence

Java persistence is maintained through the Hibernate object/relational query service. Hibernate creates Java persistence classes that conform to object-oriented association, inheritance, composition and collections. The MTS uses Hibernate tools to reverse engineer data model structures to create domain model classes, Hibernate mapping files, and entity beans. Using the Hibernate console configuration, an entity model can be created from a corresponding data model. Once created, Hibernate code can be generated from the entity model.

To generate Hibernate code:

Select Domain code and the hibernate.xml.cfg options in the code generation panel.

- Under the package explorer view under web content open the hibernate.reveng.xml file to display the Hibernate mappings.
- Select the tables that you would like to create Java entities.
- All generated classes must belong to the org.dhhs.mtcse.entity package.
- Name the class names appropriately.
- Select the key column defined in the database. After completing all the tab values, click on Hibernate Code Generation from the toolbar.
- Once complete click “Run” and generate the selected table. To view the object relations select the entity model.
For example: Enter org.dhhs.mtcse.entity.Role as the Class Name if the name of the table is mtcse_role.

To add Primary Key and the Generator for the Key Column, click “Add Primary Key”.

For example: Enter org.dhhs.mtcse.entity.Role as the Class Name if the name of the table is mtcse_role.

4.5.2 Package Structure (Java Classes)

org.dhhs.mtcse.beans – This package contains Java Beans that represent Flex Value Objects and used in Web Service Request/Response Data.

To represent a User detail object, the Java Class will be named as UserVOType.

org.dhhs.mtcse.entity – This package contains the Hibernate reverse engineering classes that represent entities

org.dhhs.mtcse.exception – This package contains the exception handling related classes

org.dhhs.mtcse.service – This package contains Java Classes that represent Web Services

To develop a UserAdministrationWebService, create a Plain Old Java Object (POJO) in this package called UserAdministrationWebService.java

Each class must have a handle to the Logger class as shown below.

    private static Logger log = LogService.getLogger(UserAdministrationWebService.class);

The following lines define accessing the Business Service Interface.

    /**
     * Get the service interface.
     *
     * @return IUserAdministrationService interface
     */
    private IUserAdministrationService getUserAdministrationService() {
        try {
            return (IUserAdministrationService)
                ServiceLocator.getService(ServiceLocator.USERADMINISTRATION_SERVICE);
        } catch (Exception e) {
            return null;
        }
    }

All methods that are defined as operations must be defined public. If a specific web service operation returns a collection, WS-I recommends using simple arrays instead of complex Java Collections.
For example, to return list of all users, return a UserVOType array

```java
public UserVOType[] getUsers() throws GenericBusinessException {
    org.dhhs.mtcse.servicelocator – This package contains the framework necessary for locating the service interface and returning it to the Web Service. For each new Service and Interface added, update the ServiceLocator class contained in this package.

- The code above defines the User Administration Service Constant as defined in the service.properties file.

/**
 * The AuthenticationServiceService name.
 */

public static final String USERADMINISTRATION_SERVICE = "USERADMINISTRATION_SERVICE";

- In addition, also add the following two definitions to the ServiceLocator class:

```java
serviceInfoMap.put(USERADMINISTRATION_SERVICE, props.getProperty(USER ADMINISTRATION_SERVICE));
```

```java
serviceInterfaceMap.put(USERADMINISTRATION_SERVICE, "org.dhhs.mtcse.session.IUserAdministrationService");
```

- **Note:** Also add an entry to the service.xml file as follows:

```xml
USERADMINISTRATION_SERVICE=org.dhhs.mtcse.session.UserAdministrationService
```

`org.dhhs.mtcse.session` – This package contains all the interfaces and class implementations for these interfaces or business services. Interface names start with prefix “I”.

- The interface defines all methods that are to be implemented by the class:

```java
public interface IUserAdministrationService {
    public User addUser(User user) throws GenericBusinessException;

    public void deleteUser(String user) throws GenericBusinessException;

    public User getUserById(String userID) throws GenericBusinessException;

    public java.util.Collection<User> getUserList() throws GenericBusinessException;

    public java.util.Collection<User> getUserList(int startIndex, int endIndex) throws GenericBusinessException;

    public int getUsersSize() throws GenericBusinessException;

    public void saveUser(User user) throws GenericBusinessException;
}
```
Now the class that implements the interface will be defined as follows:

```java
public class UserAdministrationService implements IUserAdministrationService, LocatableService {
    ...
    ...
    ...
    ...
    ...
    ...

    /* (non-Javadoc)
     * @see org.dhhs.mtcse.session.IUserAdministrationService#addUser(org.dhhs.mtcse.entity.User)
     */
    public User addUser(User user) throws GenericBusinessException {
        org.dhhs.mtcse.util.HibernateQueryHelper hibernateTemplate = new org.dhhs.mtcse.util.HibernateQueryHelper();
        log.debug("Adding a New User");
        try {
            hibernateTemplate.save(user);
        } finally {
            log.debug("finished addUser(User user)");
        }
        return null;
    }
}
```

`org.dhhs.mtcse.util` – This package contains any utility classes including the Hibernate supporting utilities.

`org.dhhs.mtcse.util.log` – This package contains the logging classes for Log4J

### 4.5.3 Web Service Generation

Although the web services have been defined as POJOs the actual web service can be generated from within the Eclipse IDE. The web services are Java classes located in the package `org.dhhs.mtcse.service`. The web service uses Apache Access as the web service runtime and will default to this value during the generation of the web services.

These POJOs can be transformed into web services by the following process:

- Select the Java program and right click for the context menu.
- Select Web Services->Create Web Service.
• Validate the server and project information and generate the web service for the target server. The process will include the generation of the WSDL file once the file is generated.
4.6 Java Conventions and Standards

The MTS J2EE Java standards given below are primarily based on standards defined in “Java Programmer Style Guidelines” published by Geotechnical Software Services available at http://geosoft.no/development/javastyle.html. These guidelines have been modified based on specific needs of the MTS project. All Java source should adhere to the prescribed standards. Programming standards and code reviews will be based on the following recommendations and requirements.

4.6.1 Naming Conventions

4.6.1.1 General Naming Conventions

<table>
<thead>
<tr>
<th>Names representing packages should be in all lower case.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example: mypackage, com.company.application.ui</td>
</tr>
</tbody>
</table>

Package naming convention used by Sun for the Java core packages. The initial package name representing the domain name must be in lower case.

<table>
<thead>
<tr>
<th>Names representing types must be nouns and written in mixed case starting with upper case.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example: Line, AudioSystem</td>
</tr>
</tbody>
</table>

Common practice in the Java development community and also the type naming convention used by Sun for the Java core packages.
### Variable names must be in mixed case starting with lower case.

**Example:** line, audioSystem

Common practice in the Java development community and also the naming convention for variables used by Sun for the Java core packages. Makes variables easy to distinguish from types, and effectively resolves potential naming collision as in the declaration `Line line;`

### Names representing constants (final variables) must be all uppercase using underscore to separate words.

**Example:** MAX_ITERATIONS, COLOR_RED

Common practice in the Java development community and also the naming convention used by Sun for the Java core packages. In general, the use of such constants should be minimized. In many cases implementing the value as a method is a better choice:

```java
int getMaxIterations() // NOT: MAX_ITERATIONS = 25
{
    return 25;
}
```

This form is both easier to read, and it ensures a uniform interface towards class values.
### Names representing methods must be verbs and written in mixed case starting with lower case.

**Example:** getName(), computeTotalWidth()

Common practice in the Java development community and also the naming convention used by Sun for the Java core packages. This is identical to variable names, but methods in Java are already distinguishable from variables by their specific form.

### Abbreviations and acronyms should not be uppercase when used as name.

**Example:**

```java
exportHtmlSource(); // NOT: exportHTMLSource();
openDvdPlayer(); // NOT: openDVDPlayer();
```

Using all uppercase for the base name will give conflicts with the naming conventions given above. A variable of this type would have to be named dVD, hTML etc. which obviously is not very readable. Another problem is illustrated in the examples above; When the name is connected to another, the readability is seriously reduced; The word following the acronym does not stand out as it should.

### Private class variables should have underscore suffix.

**Example:**

```java
class Person { private String name_; ... }
```

Apart from its name and its type, the scope of a variable is its most important feature. Indicating class scope by using underscore makes it easy to distinguish class variables from local scratch variables. This is important because class variables are considered to have higher
Private class variables should have underscore suffix.

Private class variables should have underscore suffix. significance than method variables, and should be treated with special care by the programmer. A side effect of the underscore naming convention is that it nicely resolves the problem of finding reasonable variable names for setter methods:

```java
void setName(String name)
{
    name_ = name;
}
```

An issue is whether the underscore should be added as a prefix or as a suffix. Both practices are commonly used, but the latter is recommended because it seems to best preserve the readability of the name.

It should be noted that scope identification in variables have been a controversial issue for quite some time. It seems, though, that this practice now is gaining acceptance and that it is becoming more and more common as a convention in the professional development community.

Generic variables should have the same name as their type.

Example:

```java
void setTopic(Topic topic)
    // NOT: void setTopic(Topic value)
    // NOT: void setTopic(Topic aTopic)
    // NOT: void setTopic(Topic t)
void connect(Database database)
    // NOT: void connect(Database db)
    // NOT: void connect(Database oracleDB)
```
Reduce complexity by reducing the number of terms and names used. Also makes it easy to deduce the type given a variable name only. If for some reason this convention doesn't seem to fit it is a strong indication that the type name is badly chosen. Non-generic variables have a role. These variables can often be named by combining role and type:

- Point startingPoint, centerPoint;
- Name loginName;

All names should be written in English.

Example:

English is the preferred language for international development.

Variables with a large scope should have long names, variables with a small scope can have short names.

Example:

Scratch variables used for temporary storage or indices are best kept short. A programmer reading such variables should be able to assume that its value is not used outside a few lines of code. Common scratch variables for integers are i, j, k, m, n and for characters c and d.
### The name of the object is implicit, and should be avoided in a method name.

**Example:**

```java
Line.getLength();  // NOT: line.getLineLength();
```

The latter might seem natural in the class declaration, but proves superfluous in use, as shown in the example.

### 4.6.1.2 Specific Naming Conventions

#### The terms get/set must be used where an attribute is accessed directly.

**Example:**

```java
employee.getName();
employee.setName(name);
matrix.getElement(2, 4);
matrix.setElement(2, 4, value);
```

Common practice in the Java community and the convention used by Sun for the Java core packages.
### is prefix should be used for boolean variables and methods.

**Example:**

isSet, isVisible, isFinished, isFound, isOpen

This is the naming convention for boolean methods and variables used by Sun for the Java core packages.

Using the *is* prefix solves a common problem of choosing bad boolean names like *status* or *flag* *isStatus* or *isFlag* simply doesn't fit, and the programmer is forced to choose more meaningful names.

Setter methods for boolean variables must have *set* prefix as in:

```java
void setFound(boolean isFound);
```

There are a few alternatives to the *is* prefix that fits better in some situations. These are *has*, *can* and *should* prefixes:

```java
boolean hasLicense();
boolean canEvaluate();
boolean shouldAbort = false;
```

### The term compute can be used in methods where something is computed.

**Example:**

```java
valueSet.computeAverage(); matrix.computeInverse()
```

Give the reader the immediate clue that this is a potential time consuming operation, and if used repeatedly, he might consider caching the result. Consistent use of the term enhances readability.
The term find can be used in methods where something is looked up.

**Example:**
vertex.findNearestVertex();
matrix.findSmallestElement();
node.findShortestPath(Node destinationNode);

Give the reader the immediate clue that this is a simple look up method with a minimum of computations involved. Consistent use of the term enhances readability.

The term initialize can be used where an object or a concept is established.

**Example:**
printer.initializeFontSet();

The American initialize should be preferred over the English initialise. The abbreviation init must be avoided.

**JFC (Java Swing) variables should be suffixed by the element type.**

**Example:**
widthScale, nameTextField, leftScrollbar, mainPanel, fileToggle, minLabel, printerDialog
Enhances readability since the name gives the user an immediate clue of the type of the variable and thereby the available resources of the object.

**Plural form should be used on names representing a collection of objects.**

**Example:**
Collection<Point> points; int[] values;

Enhances readability since the name gives the user an immediate clue of the type of the variable and the operations that can be performed on its elements.

**n prefix should be used for variables representing a number of objects.**

**Example:**
nPoints, nLines

The notation is taken from mathematics where it is an established convention for indicating a number of objects.

Note that Sun use *num* prefix in the core Java packages for such variables. This is probably meant as an abbreviation of *number of*, but as it looks more like *number* it makes the variable name strange and misleading. If "number of" is the preferred phrase, *numberOf* prefix can be used instead of just *n. num* prefix must not be used.
<table>
<thead>
<tr>
<th>No suffix should be used for variables representing an entity number.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Example:</strong></td>
</tr>
<tr>
<td>tableNo, employeeNo</td>
</tr>
</tbody>
</table>

The notation is taken from mathematics where it is an established convention for indicating an entity number. An elegant alternative is to prefix such variables with an $i$: iTable, iEmployee. This effectively makes them *named* iterators.

<table>
<thead>
<tr>
<th>Iterator variables should be called $i, j, k$ etc.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Example:</strong></td>
</tr>
<tr>
<td>For (Iterator i = points.iterator(); i.hasNext(); ) {</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

The notation is taken from mathematics where it is an established convention for indicating iterators. Variables named $j$, $k$ etc. should be used for nested loops only.
Complement names must be used for complement entities [1].

**Example:**
get/set, add/remove, create/destroy, start/stop, insert/delete, increment/decrement, old/new, begin/end, first/last, up/down, min/max, next/previous, old/new, open/close, show/hide, suspend/resume, etc.

Reduce complexity by symmetry.
### Abbreviations in names should be avoided.

**Example:**

```java
computeAverage(); // NOT: compAvg();
ActionEvent event; // NOT: ActionEvent e;
catch (Exception exception) { // NOT: catch (Exception e) {
```

There are two types of words to consider. Common words listed in a language dictionary must never be abbreviated. Never write:

- `cmd` instead of `command`
- `comp` instead of `compute`
- `cp` instead of `copy`
- `e` instead of `exception`
- `init` instead of `initialize`
- `pt` instead of `point`

etc.

Then there are domain specific phrases that are more naturally known through their acronym or abbreviations. These phrases should be kept abbreviated. Never write:

- `HypertextMarkupLanguage` instead of `html`
- `CentralProcessingUnit` instead of `cpu`
- `PriceEarningRatio` instead of `pe`

etc.
**Negated boolean variable names must be avoided.**

**Example:**

```java
bool isError; // NOT: isNoError
bool isFound; // NOT: isNotFound
```

The problem arise when the logical not operator is used and double negative arises. It is not immediately apparent what !isNotError means.

---

**Associated constants (final variables) should be prefixed by a common type name.**

**Example:**

```java
final int COLOR_RED = 1;
final int COLOR_GREEN = 2;
final int COLOR_BLUE = 3;
```
This indicates that the constants belong together, and what concept the constants represent.
An alternative to this approach is to put the constants inside an interface effectively prefixing their names with the name of the interface:

```java
interface Color {
    final int RED   = 1;
    final int GREEN = 2;
    final int BLUE  = 3;
}
```

Exception classes should be suffixed with `Exception`.

**Example:**
Class `AccessException` extends `Exception` {
}

Exception classes are really not part of the main design of the program, and naming them like this makes them stand out relative to the other classes. This standard is followed by Sun in the basic Java library.

Default interface implementations can be prefixed by `Default`.

**Example:**
Class `DefaultTableCellRenderer` implements `TableCellRenderer` {
}
It is not uncommon to create a simplistic class implementation of an interface providing default behavior to the interface methods. The convention of prefixing these classes by default has been adopted by Sun for the Java library.

**Singleton classes should return their sole instance through method getInstance.**

**Example:**

Class UnitManager

```java
private final static UnitManager instance_ = new UnitManager();
private UnitManager()
{
...
}
public static UnitManager getInstance() // NOT: get() or instance() or unitManager() etc.
{ return instance_; }
```}

Common practice in the Java community though not consistently followed by Sun in the JDK. The above layout is the preferred pattern.
**Classes that creates instances on behalf of others (factories) can do so through method new[ClassName].**

**Example:**

```java
Class PointFactory
{
    public Point newPoint(...)
    {
        ...
    }
}
```

Indicates that the instance is created by new inside the factory method and that the construct is a controlled replacement of new Point().

**Functions (methods returning an object) should be named after what they return and procedures (void methods) after what they do.**

Increases readability. Makes it clear what the unit should do and especially all the things it is not supposed to do. This again makes it easier to keep the code clean of side effects.
4.6.2 Java Source Files

**Java source files should have the extension .java.**

**Example:**
Point.java

Enforced by the Java tools.

**Classes should be declared in individual files with the file name matching the class name. Secondary private classes can be declared as inner classes and reside in the file of the class they belong to.**

**Example:**

Enforced by the Java tools.
### File content must be kept within 80 columns.

**Example:**

80 columns is the common dimension for editors, terminal emulators, printers and debuggers, and files that are shared between several developers should keep within these constraints. It improves readability when unintentional line breaks are avoided when passing a file between programmers.

### Special characters like TAB and page break must be avoided.

**Example:**

These characters are bound to cause problems for editors, printers, terminal emulators or debuggers when used in a multi-programmer, multi-platform environment.
The incompleteness of split lines must be made obvious.

Example:

totalSum = a + b + c + d + e;
method(param1, param2, param3);
setText("Long line split" + "into two parts.");
for (int tableNo = 0; tableNo < nTables; tableNo += tableStep) {
  ...
}

Split lines occurs when a statement exceed the 80 column limit given above. It is difficult to give rigid rules for how lines should be split, but the examples above should give a general hint.

In general:
- Break after a comma.
- Break after an operator.
- Align the new line with the beginning of the expression on the previous line.
4.6.3 Program Statements

4.6.3.1 Package and Import Statements

<table>
<thead>
<tr>
<th>The package statement must be the first statement of the file. All files should belong to a specific package.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example:</td>
</tr>
</tbody>
</table>

The package statement location is enforced by the Java language. Letting all files belong to an actual (rather than the Java default) package enforces Java language object oriented programming techniques.

<table>
<thead>
<tr>
<th>The import statements must follow the package statement. Import statements should be sorted with the most fundamental packages first, and grouped with associated packages together and one blank line between groups.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example:</td>
</tr>
</tbody>
</table>

```java
import java.io.IOException;
import java.net.URL;
import java.rmi.RmiServer;
import java.rmi.server.Server;
import javax.swing.JPanel;
import javax.swing.event.ActionEvent;
import org.linux.apache.server.SoapServer;
```
The import statement location is enforced by the Java language. The sorting makes it simple to browse the list when there are many imports, and it makes it easy to determine the dependencies of the present package. The grouping reduces complexity by collapsing related information into a common unit.

### Imported classes should always be listed explicitly.

**Example:**

```java
import java.util.List;  // NOT: import java.util.*;
import java.util.ArrayList;
import java.util.HashSet;
```

Importing classes explicitly gives an excellent documentation value for the class at hand and makes the class easier to comprehend and maintain.

Appropriate tools should be used in order to always keep the import list minimal and up to date.
4.6.3.2 Classes and Interfaces

Class and Interface declarations should be organized in the following manner:
1. Class/Interface documentation.
2. Class or interface statement.
3. Class (static) variables in the order public, protected, package (no access modifier), private.
4. Instance variables in the order public, protected, package (no access modifier), private.
5. Constructors.

Methods (no specific order).

Example:

Reduce complexity by making the location of each class element predictable.

4.6.3.3 Methods

Method modifiers should be given in the following order:
<access> static abstract synchronized <unusual> final native

The <access> modifier (if present) must be the first modifier.

Example:

public static double square(double a); // NOT: static public double square(double a);
<access> is public, protected or private while <unusual> includes volatile and transient. The most important lesson here is to keep the access modifier as the first modifier. Of the possible modifiers, this is by far the most important, and it must stand out in the method declaration. For the other modifiers, the order is less important, but it makes sense to have a fixed convention.

4.6.3.4 Types

Type conversions must always be done explicitly. Never rely on implicit type conversion.

Example:
floatValue = (int) intValue; // NOT: floatValue = intValue;

By this, the programmer indicates that he is aware of the different types involved and that the mix is intentional.

Array specifiers must be attached to the type not the variable.

Example:
int[] a = new int[20]; // NOT: int a[] = new int[20]

The array-ness is a feature of the base type, not the variable. It is not known why Sun allows both forms.
### 4.6.3.5 Variables

<table>
<thead>
<tr>
<th><strong>Variables should be initialized where they are declared and they should be declared in the smallest scope possible.</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Example:</strong></td>
</tr>
<tr>
<td>This ensures that variables are valid at any time. Sometimes it is impossible to initialize a variable to a valid value where it is declared. In these cases it should be left uninitialized rather than initialized to some phony value.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Variables must never have dual meaning.</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Example:</strong></td>
</tr>
<tr>
<td>Enhances readability by ensuring all concepts are represented uniquely. Reduce chance of error by side effects.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Class variables should never be declared public.</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Example:</strong></td>
</tr>
<tr>
<td>The concept of Java information hiding and encapsulation is violated by public variables. Use private variables and access functions instead. One exception to this rule is when the class is essentially a data structure, with no behavior (equivalent to a C++ struct). In this case it is appropriate to make the class' instance variables public [2].</td>
</tr>
</tbody>
</table>
Arrays should be declared with their brackets next to the type.

**Example:**

double[] vertex;  // NOT: double vertex[];
int[] count;  // NOT: int count[];

```java
public static void main(String[] arguments)
public double[] computeVertex()
```

The reason for is twofold. First, the array-ness is a feature of the class, not the variable. Second, when returning an array from a method, it is not possible to have the brackets with other than the type (as shown in the last example).

---

Variables should be kept alive for as short a time as possible.

**Example:**

Keeping the operations on a variable within a small scope, it is easier to control the effects and side effects of the variable.
### 4.6.3.6 Loops

**Only loop control statements must be included in the for() construction.**

**Example:**

```c
sum = 0; // NOT: for (i = 0, sum = 0; i < 100; i++)
for (i = 0; i < 100; i++) sum += value[i];
sum += value[i];
```

Increase maintainability and readability. Make a clear distinction of what controls and what is contained in the loop.

**Loop variables should be initialized immediately before the loop.**

**Example:**

```c
isDone = false;  // NOT: bool isDone = false;
while (!isDone) {
  // :
  // while (!isDone) {
  // :
  // }
  // }
```
The use of do-while loops can be avoided.

Example:

Do-while loops are less readable than ordinary while loops and for loops since the conditional is at the bottom of the loop. The reader must scan the entire loop in order to understand the scope of the loop.
In addition, do-while loops are not needed. Any do-while loop can easily be rewritten into a while loop or a for loop. Reducing the number of constructs used enhances readability.

The use of break and continue in loops should be avoided.

Example:

These statements should only be used if they prove to give higher readability than their structured counterparts.
4.6.3.7  Conditionals

<table>
<thead>
<tr>
<th>Complex conditional expressions must be avoided. Introduce temporary boolean variables instead [1].</th>
</tr>
</thead>
</table>

**Example:**

```c
bool isFinished = (elementNo < 0) || (elementNo > maxElement);
bool isRepeatedEntry = elementNo == lastElement;
if (isFinished || isRepeatedEntry) {
  // NOT: if ((elementNo < 0) || (elementNo > maxElement)|| elementNo == lastElement) {
  :
  }
}
```

By assigning Boolean variables to expressions, the program gets automatic documentation. The construction will be easier to read, debug and maintain.
The nominal case should be put in the if-part and the exception in the else-part of an if statement [1].

**Example:**

```java
boolean isOk = readFile(fileName); if (isOk) {
  :
} else {
  :
}
```

Makes sure that the exceptions do not obscure the normal path of execution. This is important for both the readability and performance.

The conditional should be put on a separate line.

**Example:**

```java
if (isDone)             // NOT: if (isDone) doCleanup();
doCleanup();
```

This is for debugging purposes. When writing on a single line, it is not apparent whether the test is really true or not.
Executable statements in conditionals must be avoided.

Example:
InputStream stream = File.open(fileName, "w");
if (stream != null) {
    :
}
// NOT: if (File.open(fileName, "w") != null) {
    :
}

Conditionals with executable statements are simply very difficult to read. This is especially true for programmers new to Java.
4.6.3.8 Miscellaneous

The use of magic numbers in the code should be avoided. Numbers other than 0 and 1 can be considered declared as named constants instead.

Example:

```java
private static final int TEAM_SIZE = 11;
:
Player[] players = new Player[TEAM_SIZE]; // NOT: Player[] players = new Player[11];
```

If the number does not have an obvious meaning by itself, the readability is enhanced by introducing a named constant instead.

Floating point constants should always be written with decimal point and at least one decimal.

Example:

```java
double total = 0.0; // NOT: double total = 0;
double speed = 3.0e8; // NOT: double speed = 3e8;
double sum;
:
sum = (a + b) * 10.0;
```
This emphasizes the different nature of integer and floating point numbers. Mathematically the two model completely different and non-compatible concepts.

Also, as in the last example above, it emphasizes the type of the assigned variable (sum) at a point in the code where this might not be evident.

**Floating point constants should always be written with a digit before the decimal point.**

**Example:**

double total = 0.5; // NOT: double total = .5;

The number and expression system in Java is borrowed from mathematics and one should adhere to mathematical conventions for syntax wherever possible. Also, 0.5 is a lot more readable than .5; There is no way it can be mixed with the integer 5.

**Static variables or methods must always be referred to through the class name and never through an instance variable.**

**Example:**

Thread.sleep(1000); // NOT: thread.sleep(1000);

This emphasize that the element references is static and independent of any particular instance. For the same reason the class name should also be included when a variable or method is accessed from within the same class.
4.6.4 Layout and Comments

4.6.4.1 Layout

<table>
<thead>
<tr>
<th>Basic indentation should be 2.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Example:</strong></td>
</tr>
<tr>
<td>for (i = 0; i &lt; nElements; i++)</td>
</tr>
<tr>
<td>a[i] = 0;</td>
</tr>
</tbody>
</table>

Indentation is used to emphasize the logical structure of the code. Indentation of 1 is too small to achieve this. Indentation larger than 4 makes deeply nested code difficult to read and increase the chance that the lines must be split. Choosing between indentation of 2, 3 and 4; 2 and 4 are the more common, and 2 chosen to reduce the chance of splitting code lines. Note that the Sun recommendation on this point is 4.
Block layout should be as illustrated in example 1 below (recommended) or example 2, and must not be as shown in example 3. Class, Interface and method blocks should use the block layout of example 2.

Example 1:
while (!done)
{
  doSomething();
  done = moreToDo();
}

Example 2:
while (!done)
{
  doSomething();
  done = moreToDo();
}

Example 3:
while (!done) {
  doSomething();
  done = moreToDo();
}

Example 3 introduces an extra indentation level which doesn't emphasize the logical structure of the code as clearly as example 1 and 2.
The class and interface declarations should have the following form:

**Example:**
class Rectangle extends Shape
implements Cloneable, Serializable {
    ...
}

This follows from the general block rule above. *Note that it is common in the Java developer community to have the opening bracket at the end of the line of the class keyword. This is not recommended.*

Method definitions should have the following form:

**Example:**
public void someMethod() throws SomeException {
    ...
}

See comment on class statements above.
The if-else class of statements should have the following form:

**Example:**
```java
if (condition){
statements;
}
if (condition){
statements;
}
else {
statements;
}
if (condition) {
statements;
}
else if (condition) {
statements;
}
else {
statements;
}
```
The if-else class of statements should have the following form:

This follows partly from the general block rule above. However, it might be discussed if an else clause should be on the same line as the closing bracket of the previous if or else clause:

```java
if (condition) {
    statements;
} else {
    statements;
}
```

This is equivalent to the Sun recommendation. The chosen approach is considered better in the way that each part of the if-else statement is written on separate lines of the file. This should make it easier to manipulate the statement, for instance when moving else clauses around.

The for statement should have the following form:

**Example:**

```java
for (initialization; condition; update) {
    statements;
}
```

This follows from the general block rule above.
An empty for statement should have the following form:

**Example:**
```
for (initialization; condition; update) 
;
```

This emphasize the fact that the for statement is empty and it makes it obvious for the reader that this is intentional.

The while statement should have the following form:

**Example:**
```
while (condition) {
statements;
}
```

This follows from the general block rule above.

The do-while statement should have the following form:

**Example:**
```
do {
 statements;
} while (condition);
```

This follows from the general block rule above.
### The switch statement should have the following form:

**Example:**
```
switch (condition) {
    case ABC :
        statements; // Fallthrough
    case DEF :
        statements;
        break;
    case XYZ :
        statements;
        break;
    default :
        statements;
        break;
}
```

This differs slightly from the Sun recommendation both in indentation and spacing. In particular, each case keyword is indented relative to the switch statement as a whole. This makes the entire switch statement stand out. Note also the extra space before the : character. The explicit Fallthrough comment should be included whenever there is a case statement without a break statement. Leaving the break out is a common error, and it must be made clear that it is intentional when it is not there.
A try-catch statement should have the following form:

Example:

```java
try {
    statements;
} catch (Exception exception) {
    statements; } try {
    statements;
} catch (Exception exception) {
    statements; } finally {
    statements;
}
```

This follows partly from the general block rule above. This form differs from the Sun recommendation in the same way as the if-else statement described above.
Single statement if-else, for or while statements can be written without brackets.

Example:
if (condition)
statement;
while (condition)
statement;
or (initialization; condition; update)
statement;

It is a common recommendation (Sun Java recommendation included) that brackets should always be used in all these cases. However, brackets are in general a language construct that groups several statements. Brackets are per definition superfluous on a single statement. A common argument against this syntax is that the code will break if an additional statement is added without also adding the brackets. In general however, code should never be written to accommodate for changes that might arise.
4.6.4.1.1 White Space

- Operators should be surrounded by a space character.
- Java reserved words should be followed by a white space.
- Commas should be followed by a white space.
- Colons should be surrounded by white space.
- Semicolons in for statements should be followed by a space character.

Example:
```
a = (b + c) * d; // NOT: a=(b+c)*d
while (true) { // NOT: while(true){
    ...
doSomething(a, b, c, d); // NOT: doSomething(a,b,c,d);
case 100 : // NOT: case 100:
    for (i = 0; i < 10; i++) { // NOT: for(i=0;i<10;i++){
        ...
```

Makes the individual components of the statements stand out and enhances readability. It is difficult to give a complete list of the
suggested use of whitespace in Java code. The examples above however should give a general idea of the intentions.

Method names can be followed by a white space when it is followed by another name.

Example:
doSomething (currentFile);
Method names can be followed by a white space when it is followed by another name.

Makes the individual names stand out. Enhances readability. When no name follows, the space can be omitted (doSomething()) since there is no doubt about the name in this case.

An alternative to this approach is to require a space after the opening parenthesis. Those that adhere to this standard usually also leave a space before the closing parentheses: doSomething( currentFile );. This do make the individual names stand out as is the intention, but the space before the closing parenthesis is rather artificial, and without this space the statement looks rather asymmetrical (doSomething( currentFile );).

Logical units within a block should be separated by one blank line.

Example:

```plaintext
// Create a new identity matrix
    Matrix4x4 matrix = new Matrix4x4();

// Precompute angles for efficiency
    double cosAngle = Math.cos(angle);
    double sinAngle = Math.sin(angle);

// Specify matrix as a rotation transformation
    matrix.setElement(1, 1, cosAngle);
    matrix.setElement(1, 2, sinAngle);
    matrix.setElement(2, 1, -sinAngle);
    matrix.setElement(2, 2, cosAngle);

// Apply rotation
    transformation.multiply(matrix);
```
Enhances readability by introducing white space between logical units. Each block is often introduced by a comment as indicated in the example above.

<table>
<thead>
<tr>
<th>Variables in declarations can be left aligned.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Example:</strong></td>
</tr>
</tbody>
</table>
| ```c
   TextFile file;
   int nPoints;
   double x, y;
```|

Enhances readability. The variables are easier to spot from the types by alignment.
Statements should be aligned wherever this enhances readability.

Example:

```java
if (a == lowValue) computeSomething();
else if (a == mediumValue) computeSomethingElse();
else if (a == highValue) computeSomethingElseYet();
value = (potential * oilDensity) / constant1 +
    (depth * waterDensity) / constant2 +
    (zCoordinateValue * gasDensity) / constant3;
minPosition = computeDistance(min, x, y, z);
averagePosition = computeDistance(average, x, y, z);
switch (phase) {
    case PHASE_OIL : text = "Oil"; break;
    case PHASE_WATER : text = "Water"; break;
    case PHASE_GAS : text = "Gas"; break;
}
```

There are a number of places in the code where white space can be included to enhance readability even if this violates common guidelines. Many of these cases have to do with code alignment. General guidelines on code alignment are difficult to give, but the examples above should give some general hints. In short, any construction that enhances readability should be allowed.
4.6.4.1.2 Comments

<table>
<thead>
<tr>
<th>Tricky code should not be commented but rewritten.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example:</td>
</tr>
<tr>
<td>In general, the use of comments should be minimized by making the code self-documenting by appropriate name choices and an explicit logical structure.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>All comments should be written in English.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example:</td>
</tr>
<tr>
<td>In an international environment English is the preferred language.</td>
</tr>
</tbody>
</table>
Javadoc comments should have the following form:

Example:

```java
/**
 * Return lateral location of the specified position.
 * If the position is unset, NaN is returned.
 * @param x X coordinate of position.
 * @param y Y coordinate of position.
 * @param zone Zone of position.
 * @return Lateral location.
 * @throws IllegalArgumentException If zone is <= 0.
 */
public double computeLocation(double x, double y, int zone)
  throws IllegalArgumentException {
  ...
}
```
### Javadoc comments should have the following form:

A readable form is important because this type of documentation is typically read more often *inside* the code than it is as processed text. Note in particular:

- The opening /** on a separate line
- Subsequent * is aligned with the first one
- Space after each *
- Empty line between description and parameter section
- Alignment of parameter descriptions
- Punctuation behind each parameter description
- No blank line between the documentation block and the method/class

Javadoc of class members can be specified on a single line as follows:

```java
/** Number of connections to this database */
private int nConnections_;```
### There should be a space after the comment identifier.

**Example:**

```java
// This is a comment  NOT: //This is a comment
/**
 * This is a javadoc
 * comment */
NOT: /**
 *This is a javadoc
 *comment
 */
```

Improves readability by making the text stand out.

### Use // for all non-Javadoc comments, including multi-line comments.

**Example:**

```java
// Comment spanning
  // more than one line.
```

Since multilevel Java commenting is not supported, using // comments ensure that it is always possible to comment out entire sections of a file using /* */ for debugging purposes etc.
### Comments should be indented relative to their position in the code [1].

**Example:**
```java
while (true) {
    // Do something
    something();
}
// NOT: while (true) {
// Do something
    something();
}
```

This is to avoid that the comments break the logical structure of the program.

### The declaration of anonymous collection variables should be followed by a comment stating the common type of the elements of the collection.

**Example:**
```java
private Vector points_; // of Point
private Set shapes_; // of Shape
```
The declaration of anonymous collection variables should be followed by a comment stating the common type of the elements of the collection.

Without the extra comment it can be hard to figure out what the collection consist of, and thereby how to treat the elements of the collection. In methods taking collection variables as input, the common type of the elements should be given in the associated JavaDoc comment.

Whenever possible one should of course qualify the collection with the type to make the comment superfluous:

```java
private Vector<Point> points_;
private Set<Shape> shapes_;```

All public classes and public and protected functions within public classes should be documented using the Java documentation (javadoc) conventions.

Example:

This makes it easy to keep up-to-date online code documentation.
4.7 Exception Handling

The exception handling framework for the MTS accommodate for error handling and modes of detection, isolation, and messaging that may occur during the operation of the application. Conditions which cause an exception during a UI session will produce application information, warnings, and errors messages to the user. On-line, background and scheduled architectures will be capable of handling failures and warnings and will allow for the requisite error messaging, logging, reporting, and administrator notification as dictated by the particular error handling process. The Exception Handling & Logging framework will be based on Java Exception Hierarchy and Logging, which is based on the Log4J Logging API.

System errors, program errors, navigation errors, and logging errors will be tracked and recorded by the application for system maintenance. Error processing will be evaluated differently based on the type of error and on the audience associated with the area in which the error occurs.

The Exception Processing Framework Use Case document details the functionality and components for exception handling within the MTS.

4.7.1 Error Messages

Error messages are either user-based or environment-based. The user error message will provide sufficient information to identify and correct the underlying problem. The environment error will need to provide detailed information on the corrective action required and possibly the current status of the application.

Error messages produced by the application will be designed to be tailored for specific usages within the application. The following cites the specific properties the message categories use within the MTS application:

**Enterprise Messages**

error. – Error messages used throughout the application.

warning. – Warning message used throughout the application.

message. – Messages used throughout the application.

**Flex Specific Messages**

flex.error. – Flex Error messages used throughout the online application.

flex.warning. – Flex Warning message used throughout the online application.

flex.message. – Flex Message used throughout the online application.

flex.panel.error. – Error messages used throughout the panel of the online application.

flex.panel.warning. – Warning messages used throughout the panel of the online application.

flex.panel.message. – Message used throughout the panel of the online application.

flex.container.panel.error. – Error messages used within a container and inside a panel in the online application.

flex.container.panel.warning. – Warning messages used within a container and inside a panel in the online application.
4.7.2 Exception Handling & Logging Standards

**Exceptions**

- Generally use the predefined exceptions types. Define new exception types only for programmatic scenarios. Introduce a new exception class to enable a programmer to take a different action in code based on the exception class.

- Do not derive user-defined exceptions from the Exception base class. Derive custom exceptions from the ApplicationException class.

- Include a localized description string in every exception. When a user message is displayed, it is derived from the description string of the exception that was thrown, rather than from the exception class.

- Use grammatically correct error messages, including ending punctuation. Each sentence in a description string of an exception should end in a period.

- Provide Exception properties for programmatic access. Include extra information in an exception (in addition to the description string) only when there is a programmatic scenario where the additional information is useful.

- When designing exception processing design patterns for the application, they should be grouped together rather than simply having them all extend Exception.

**Logging**

- Obtain a handle to the logging framework and simply call the log method providing the severity level for this message like “Debug”, “Info”, “Warning”, “Error”, “Fatal”.

- Developers using the logging framework to log messages for traceability, use the framework’s addUserVariable(String, Object o) method to pass variables that will help diagnosing and grouping messages within a context.

- The framework must support extended functionality to provide capabilities for logging this message either synchronously or asynchronously and if database persistence is chosen during deploy time, these log messages will be persisted to a data store.
4.8 COTS Packages

The following table summarizes the various software components including commercial-of-the-shelf (COTS) packages used in MTS.

<table>
<thead>
<tr>
<th>COTS module or application</th>
<th>Reused COTS Template</th>
<th>Benefits List</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Server</td>
<td>JBoss Application Server</td>
<td>Provides a highly scalable, highly available controlled environment. Tight integration into the Eclipse IDE.</td>
</tr>
<tr>
<td>Workflow</td>
<td>jBPM</td>
<td>Ensures data integrity in a collaborative environment.</td>
</tr>
<tr>
<td>Reporting</td>
<td>Jasper Reports</td>
<td></td>
</tr>
<tr>
<td>Apache Commons Logging</td>
<td>Apache open source</td>
<td>Industry standard mechanism to log informational, debug and error messages. Fine grained control on logging by severity, package, class, or method.</td>
</tr>
<tr>
<td>MySQL</td>
<td>Database Server</td>
<td></td>
</tr>
</tbody>
</table>

4.9 Components

The respective use case document describes all the components including interface and reusable components. Use case documentation details the following for all impacted components:

- **J2EE Components** – Listed by Components Name with a Type and Description definition.
- **User Interface Components** – Listed by Business Component Name with a Design Pattern and Component Action definition. It also lists the detail of the UI Component to include navigation and a profile or each panel (.mxml) (and panel image).

A summary of each use case can be found in Section 4, Detail Design. The following Eclipse projects contain the packages and modules related to each component defined in the Use Case documentation:

- MTS_App.
- MTS_EJBApp.
- MTS_WebApp.
4.9.1 Component ID and Component Name
Refer to the respective Eclipse project for a listing of components, packages, and modules contained within each project.

4.10 Interfaces
The MTS System Architecture document describes component interfaces.
5 Detailed Design

This section contains an overview of the detailed design for MTS requirements. This detailed design refines the high level design presented in the MTS General System Design document. MTS development will consist of the following tracks:

- **J2EE** – Presentation and application layer. The only business rules within this layer are related to data presentation and preservation and enforcement of data integrity.
- **Reports** – All user reports.
- **Database** – Data Persistence.

MTS is comprised of twelve (12) functional areas designed to support the various operational areas of child support enforcement for tribal organizations. All design elements of MTS business and system functionality is captured in use case specification documentation. This documentation describes the various usage scenarios, focusing on a specific application area. It also contains descriptions and diagrams of J2EE data components which comprise the application layer of the MTS design. All use cases which encompass interactive and background processes of the MTS design are presented in this section, categorized as follows:

- Case Management.
- Locate Management.
- Paternity Establishment.
- Order Establishment.
- Enforcement.
- Financial Management.
- Application Management.
- Document Generation.
- Security & Administration.
- Common Functions.
- Third Party Maintenance & Configuration.
- Reporting.

Each Use Case has a detailed design document including the data model for all schemas. Individual Use Case documents are available from OCSE.
Exhibit 5-0: MTS Design Functional Areas
5.1 Case Management

The Case Management component supports all defined requirements for case intake and update processing. It provides an access mechanism utilized by the caseworker to enter applications for child support services and set up child support cases, update case information, perform referrals and third party searches, among other functions.

Exhibit 5-1 diagrams the use cases which comprise the functionality of Case Management, as required by the General System Design and System Requirements Specification documents:

- Case Consolidation.
- Case Information Request.
- Case Intake.
- Case Management.
- Case Quick Locate.
- Case Referral Process.
- Participant Demographics.
- Participant Income & Assets.
- Participant Address.

Exhibit 5-1: Case Management Use Cases
5.1.1 MTS Use Case: Case Consolidation

Summary

This use case is comprised of processing that supports caseworkers in managing and consolidating duplicate child support cases. When the caseworker decides to consolidate two child support cases, the system will support the caseworker in selecting two cases for a side-by-side comparison and consolidating the cases.

Approach

Caseworkers access UI panels to perform the activities of case consolidation. The system only allows the authorized caseworker or supervisor to consolidate duplicate cases. The caseworker can only update his/her assigned cases while the caseworker supervisor can manage any case.

Caseworkers must perform a generic case search and select two cases for comparison and consolidation. The panel displays a case overview including participants on the case. The caseworker indicates one case as a duplicate case, wherein its status is then set to Closed.

The Case Consolidation module generates documents and events as defined by the rules-based logic described within Section 2, Software Design of the Use Case document.

This module creates, reads, updates, and/or deletes data from MTS tables as depicted in Section 3, Database of the Use Case Document.

Conclusion:

The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case: Case</td>
<td>MTS_UC_Case</td>
<td>Alfresco System -&gt; MTS -&gt; Design – Detail Design</td>
</tr>
<tr>
<td>Consolidation</td>
<td>Consolidation.doc</td>
<td></td>
</tr>
</tbody>
</table>
5.1.2 MTS Use Case: Case Information Request

**Summary**

This use case is comprised of processing that supports caseworkers with information requests from other entities. It provides for generation of letters and ticklers as part of the Case Intake and Update, Order Evaluation, and Desktop Review business processes.

**Approach**

Caseworkers access UI panels to perform the activities of case information requests. The panels allow caseworkers to produce a Request for Additional Information Letter (#77) which may be addressed to the referring jurisdiction, the referring agency, and/or to one or more participants in the case. Once a letter is generated, the module creates an Information Request Tickler for monitoring its disposition and follow-up.

The Case Information Request implementation uses the DocumentGenerationWebService to create the Request for Additional Information Letter (#77) and TicklerManagerWebService to create the Information Request Tickler.

This module generates ticklers and documents as defined by the rules-based logic described within Section 2, Software Design of the Use Case document.

The module creates, reads, updates, and/or deletes data from MTS tables as depicted in Section 3, Database of the Use Case Document.

**Conclusion:**

The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case: Case Information Request</td>
<td>MTS_UC_Case Information Request.doc</td>
<td>Alfresco System -&gt; MTS -&gt; Design – Detail Design</td>
</tr>
</tbody>
</table>
5.1.3 MTS Use Case: Case Intake

Summary
This use case encompasses all the processes of MTS case intake functionality, including case creation and assessment to determine the next appropriate action on a case.

Approach
Caseworkers create a new case through creation of new participants and/or matching existing participants. The system assists in determining jurisdiction, handling Temporary Assistance for Needy Families (TANF) referrals, request additional case related information and support quick locate requests. The system will automatically assign unique ID’s to each new participant and case created. During the case intake process, the system automatically assigns the case to a caseworker based on pre-defined rules and creates the necessary documents and ticklers, and “New Case” history event.

The Case Intake module creates, reads, updates, and/or deletes data from the MTS data stores as depicted in Section 3, Database of the Case Intake Use Case Document.

Conclusion:
The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case: Case Intake</td>
<td>MTS_UC_Case Intake.doc</td>
<td>Alfresco System -&gt; MTS -&gt; Design – Detail Design</td>
</tr>
</tbody>
</table>
5.1.4 MTS Use Case: Case Management

Summary
This use case is comprised of processing to support caseworkers in managing the numerous actions affecting the status or organization of an established case. It allows the worker to redirect payments, consolidate a case, transfer a case, and closing a case. It also monitors responses to actions on a case and determines the next action that should be taken on a case.

Approach
Caseworkers navigate through a series of UI panels that display a summary of information on a selected case and its participants. This process allows workers to perform the following actions:

- View case participants and their demographics.
- View Case Events.
- View and Manage Case Notes.
- View and Manage Ticklers.
- View and Manage Service of Process.
- View and Print Documents.
- View and Manage Hearings.

The Case Management module generates tickler, documents and events as defined by the rules-based logic described within Section 2, Software Design of the Use Case document.

This module creates, reads, updates, and/or deletes data from MTS tables as depicted in Section 3, Database of the Use Case Document.

Conclusion:
The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case: Case Management</td>
<td>MTS_UC_Case Management</td>
<td>Alfresco System -&gt; MTS -&gt; Design – Detail Design</td>
</tr>
</tbody>
</table>
5.1.5 MTS Use Case: Case Quick Locate

Summary
This use case is comprised of processing that supports caseworkers with quick-locate requests from other jurisdictions. It provides for quick-locate processing for locate-only referrals from a state IV-D agency or another tribe’s IV-D agency (Quick Locate) without establishing a case.

Approach
Caseworkers access UI panels to perform the activities of quick locate. The panels allow caseworkers to search for a referent and to generate Quick Locate Response Letters (#277) either with the referent information or with a request for further information.

Caseworkers perform searches based on the following key data elements:

- Participant Name.
- Participant Social Security number (SSN).
- Participant Date of Birth (DOB).

The module interfaces with the Member Match function to search in the participant database to match the person based on the data entered. It returns any matching participant information for display and performs one of the following actions based on the search outcome:

- If the Member Match function successfully finds the participant, the module generates a Quick Locate Response Letter (#277) with the participant’s name and address.
- If the member match function cannot find the participant and search was based on partial data entered by the caseworker, the module generates a Quick Locate Response Letter (#277) with a message “Insufficient Information Provided”, to the referring jurisdiction requesting further information. The system will prompt the caseworker to set a Foreign Referral Tickler.
- If the Member Match function cannot find the participant, the module creates a new participant (saves the data in the database) and generates a Quick Locate Response Letter (#277) with the message “Participant/Referent Not Found”.

The Case Quick Locate module generates ticklers and documents as defined by the rules-based logic described within Section 2, Software Design of the Use Case document.

This module creates, reads, updates, and/or deletes data from MTS tables as depicted in Section 3, Database of the Use Case Document.

Conclusion:
The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case: Case Quick Locate</td>
<td>MTS_UC_Case Quick Locate.doc</td>
<td>Alfresco System -&gt; MTS -&gt; Design – Detail Design</td>
</tr>
</tbody>
</table>
5.1.6 MTS Use Case: Case Referral Process

Summary
The case referral process operates within the Case Intake and Update Process as a scheduled background component that executes at pre-defined intervals. This component handles processing related to Custodial Parent (CP) cooperation in TANF cases. This component interfaces with other functional areas for referral processing using the business process model framework.

Approach
This process supports the user in generating documents and setting statuses related to CP cooperation in TANF cases. The case referral process can be invoked from the following functions:

- Case Intake.
- Paternity Interview.
- Case Transfer.

This module will perform the following actions:

1. If the case is TANF and CP has cooperated, the module generates a Notice of Cooperative Status (#15), a Welcome Letter – NCP (#12), and a Welcome Letter - CP (#469).
2. If the case is TANF case and CP has not cooperated and Good Cause has not been established, the module generates a Non-Cooperation Warning Letter (#14) and sets a Non-Cooperation Tickler.
3. If the case is TANF case, the CP has not cooperated, and Good Cause has been established, the module sets the Case Status to “Suspended” and generates a Notice of Cooperative Status (#15), and sets a Good Cause Tickler.
4. Any change in cooperative status causes the module to create a Cooperative Status Event.

The Case Referral Process module generates ticklers, documents, and events as defined by the rules-based logic described within Section 2, Software Design of the Use Case document.

This module creates, reads, updates, and/or deletes data from MTS tables as depicted in Section 3, Database of the Use Case Document.

The specified MTS applications invoke the case referral process as part of the business process modeling framework within the system. The business process template is initiated within the application that performs all the necessary actions and generates the required documents.

Conclusion:
The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case: Case Referral Process</td>
<td>MTS_UC_Case Referral Process.doc</td>
<td>Alfresco System -&gt; MTS -&gt; Design – Detail Design</td>
</tr>
</tbody>
</table>
5.1.7 MTS Use Case: Participant Demographics

Summary
This use case is comprised of processing that supports caseworkers in managing participant personal data details during case setup and case maintenance.

Approach
Caseworkers access UI panels to perform the activities of participant demographics maintenance. The panels allow caseworkers to add a participant, view, enter or modify participant personal data. The system automatically assigns a new unique identifier to each participant added to the system. The Demographics panel, when used in conjunction with the Member Match Search popup, will prevent creating duplicate participants in the system. The Demographics panel will allow the caseworker to enter data for an NCP, CP, or a child.

This process allows workers to perform the following actions:

- Add/Update Participant Demographics.
- Manage Phone Numbers.
- Manage Email Addresses.
- Maintain Addresses.
- Maintain Earned Income Information.
- Maintain Assets Information.
- Maintain Health Insurance Information.

The Participant Demographics module creates, reads, updates, and/or deletes data from MTS tables as depicted in Section 3, Database of the Use Case Document.

Conclusion:
The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case: Participant Demographics</td>
<td>MTS_UC_Participant Demographics.doc</td>
<td>Alfresco System -&gt; MTS -&gt; Design – Detail Design</td>
</tr>
</tbody>
</table>
5.1.8 MTS Use Case: Participant Income and Assets

Summary
This use case is comprised of processing that supports caseworkers in managing participant earned and unearned income data including multiple sources and amounts. The system also allows entry and maintenance of vehicle data and accounts in financial institutions.

Approach
Caseworkers access UI panels to perform the activities of participant income and assets maintenance. The panels allow caseworkers to view, add, or modify participant income and assets. The system provides the capability for entry of multiple sources of income and supports entry of participant assets. Caseworkers may view lists of earned and unearned income sources and a list of available assets. The panels provide a mechanism to search the appropriate third party tables using a generic search screen and associate a particular income or asset entry with the selected third party entry.

This process allows workers to perform the following actions:

- View/Add/Modify Participant Earned Income.
- View/Add/Modify Participant Unearned Income.
- View/Manage Participant Assets:
  - Add/Modify Bank Account.
  - Add/Modify Vehicle.

The Participant Income and Assets module creates, reads, updates, and/or deletes data from MTS tables as depicted in Section 3, Database of the Use Case Document.

Conclusion:
The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case: Participant Income</td>
<td>MTS_UC_Participant Income and Assets.doc</td>
<td>Alfresco System -&gt; MTS -&gt; Design</td>
</tr>
<tr>
<td>and Assets</td>
<td></td>
<td>– Detail Design</td>
</tr>
</tbody>
</table>
5.1.9 MTS Use Case: Participant Address

**Summary**

This use case is comprised of processing that supports caseworkers in managing all types of addresses for participants and perform the necessary business functions when address changes are performed.

**Approach**

Caseworkers access UI panels to perform the activities of participant address maintenance. The panels allow caseworkers to view, add, or modify participant residential and mailing addresses. The system also provides the capability for multiple instances of address information for participants.

For each case and participant, the system will suppress the address to be printed on documents if the Family Violence Indicator is set. Also, the system will ensure that only the caseworker or the caseworker supervisor can view the participant address. If the CP has requested that access to his or her address should be restricted, the system will record this through the Address Suppression Indicator. If the Address Suppression Indicator is set for a case, participant address information will not be printed on the documents.

The Participant Address Maintenance module generates ticklers, documents, and events as defined by the rules-based logic described within Section 2, Software Design of the Use Case document.

This module creates, reads, updates, and/or deletes data from MTS tables as depicted in Section 3, Database of the Use Case Document.

**Conclusion:**

The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case: Participant Address</td>
<td>MTS_UC_Participant Address</td>
<td>Alfresco System -&gt; MTS -&gt;</td>
</tr>
<tr>
<td>Maintenance</td>
<td>Maintenance.doc</td>
<td>Design – Detail Design</td>
</tr>
</tbody>
</table>

The table shows the track, document name, and location for the referenced document.
5.2 Locate Management

The locate management component of the MTS application will support all defined requirements for participant location efforts, including outbound letter generation to state, federal and private agencies.

Exhibit 5-2 diagrams the use case which comprises the functionality of Locate Management, as required by the GSD and SRS document 4.2.1 Locate.

Exhibit 5-2: Locate Management Use Case
5.2.1 MTS Use Case: Locate

Summary
This use case is comprised of processing that supports caseworkers in locating participants, verifying addresses, locating employers, and verifying employment. It will automatically monitor the response time to outgoing documents and monitor the amount of time the case has spent in the locate function. The system will also provide ticklers to the caseworker.

Approach
Caseworkers access UI panels to perform the activities of locate. The panels allow caseworkers to enter participant address, perform address verification, enter employer address, and perform employment verification.

Caseworkers may perform the following actions to invoke locate processes within this module:

- **Participant Address**: Caseworkers can view and enter participants address and case notes data (automatically search the address). The module generates events as required. The module produces Locate Letters, Child Support Enforcement Transmittal #1 (#248), and the Tribal Enrollment Verification Form (#73), and creates a Locate Letter Tickler for all the letters sent.

- **Address Verification**: Module generates a Postmaster Letter (#468) and sets a Postmaster Letter Tickler when a new, changed, or unverified mailing address is entered for a participant and allows the caseworker to mark addresses as verified.

- **Employer Address**: Caseworkers can view and enter information on employers of participants and case notes (automatically search the employer) and enter instances of successful manual locate attempts.

- **Employment Verification**: Caseworkers can view and enter employment status information, generate an Employment Verification Letter (#6), generate an Employment Letter Tickler, and mark employment as verified.

The Locate Module generates ticklers, documents, and events as defined by the rules-based logic described within Section 2, Software Design of the Use Case document.

This module creates, reads, updates, and/or deletes data from MTS tables as depicted in Section 3, Database of the Use Case Document.

The locate design implementation uses the LocateWebService for this module as well as the DocumentGenerationWebService to generate documents and the TicklerManagerWebService to create ticklers.

Conclusion:
The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case: Locate</td>
<td>MTS_UC_Locate.doc</td>
<td>Alfresco System -&gt; MTS -&gt; Design – Detail Design</td>
</tr>
</tbody>
</table>
5.3 Paternity

The Paternity component will automatically initiate and track paternity establishment and generate documents related to the establishment of paternity. It will allow workers to record and maintain data concerning facilitation, stipulation, and genetic tests, among other data. It will automatically generate completed legal documents and notices and initiate service of process. It will document unsuccessful attempts to serve process on located NCPs.

Exhibit 5-3 diagrams the use cases which comprise the functionality of paternity, as required by the GSD and SRS documents:

- Paternity Data Entry.
- Paternity Documents.
- Paternity Facilitation.
- Paternity Genetic Testing.
- Paternity Stipulation.

Exhibit 5-3: Paternity Use Cases
5.3.1 MTS Use Case: Paternity Data Entry

Summary

This use case is comprised of processing to support caseworkers with entry of data related to paternity and the generation of documents related to paternity interviews and appointments.

Approach

Caseworkers access UI panels to perform the activities of paternity data entry. The panels allow caseworkers to capture paternity information concerning the child, generate appointment letters and schedule interviews.

The panels allow caseworkers to view, add or modify the following information:

- Current Spouse Name (CP and NCP).
- Marital Status Data.
- Sexual Relationship Data.
- Child Birth Data.
- Child Father’s name.
- Paternity Data.
- Stipulation Data.

Paternity data that is added or modified through the panels generates a Paternity Information Event.

This module also supports the caseworker in the scheduling of a Paternity Interview.

Using the common Document Generation Interface, caseworkers can select and generate the Paternity Interview Letter – First (#276) and Paternity Interview Letter – Second (#278) to prompt the CP to schedule a Paternity Interview. This module sets the corresponding ticklers (Paternity Interview One Tickler and Paternity Interview Two Tickler) through the common Tickler Interface.

The caseworker may schedule the Paternity Interview appointment using the common Calendar Function. The caseworker selects and generates the Appointment Letter – Paternity – CP (#492).

After a Paternity Interview has been conducted, the module allows recording of a Paternity Interview Event.

The Paternity Data Entry module generates ticklers, documents, and events as defined by the rules-based logic described within Section 2, Software Design of the Use Case document.

It creates, reads, updates, and/or deletes data from MTS tables as depicted in Section 3, Database of the Use Case Document.

The paternity data entry design implementation uses the PaternityWebService for this module as well as the DocumentGenerationWebService to generate documents and the TicklerManagerWebService to create ticklers.
Conclusion:
The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case: Paternity Data Entry</td>
<td>MTS_UC_Paternity Data Entry.doc</td>
<td>Alfresco System -&gt; MTS -&gt; Design – Detail Design</td>
</tr>
</tbody>
</table>

5.3.2 MTS Use Case: Paternity Documents

Summary
This use case is comprised of processing that supports caseworkers with generation of documents to support the court process for making a finding of paternity.

Approach
Caseworkers access UI panels to perform the activities of paternity document generation. They use the common Document Generation Interface to generate facilitation documents and common Tickler Interface to generate the related ticklers.

This module supports the caseworker generation of the following paternity documents:

- Letter to Establish Paternity and Child Support (#112).
- Petition to Establish Paternity and Child Support (#151).
- Paternity and Child Support Order (#498).
- Petition to Establish Paternity (#289).
- Paternity Order (#293).
- Summons – Paternity (#153).
- Motion for Genetic Testing (#290).
- Order for Genetic Testing (#543).
- Motion to Dismiss (#491).
- Order to Dismiss (#490).
- Motion for Continuance (#171).
- Order for Continuance (#544).
- Voluntary Service Acceptance Letter (#300).
- Affidavit of Mailing (#508).
The Paternity Documents module generates documents as defined by the rules-based logic described within Section 2, Software Design of the Use Case document.

It reads data from MTS tables as depicted in Section 3, Database of the Use Case Document.

The paternity documents design implementation uses the DocumentGenerationWebService to generate documents.

5.3.3 MTS Use Case: Paternity Facilitation

**Summary**

This use case is comprised of processing that supports caseworkers with generation of documents to encourage participants to voluntarily acknowledge paternity and their child support obligation. This module also monitors responses to these letters and supports the scheduling of facilitation meetings with participants.

**Approach**

Caseworkers access UI panels to perform the activities of paternity facilitation. The panels allow caseworkers to select next appropriate facilitation action and use the common Document Generation Interface to generate facilitation documents and common Tickler Interface to generate the related ticklers.

This module supports the caseworker with the following:

- Generation of a Voluntary Acknowledgement of Paternity Letter – First (#287). The module sets a VAP Letter One Tickler when this letter is generated.
- Generation of a Voluntary Acknowledgement of Paternity Letter – Second (#513). The module sets a VAP Letter Two Tickler when this letter is generated.
- Scheduling of a voluntary genetic test through the common Calendar Function and the generation of a Voluntary Genetic Test Request (#28).
- Scheduling of a facilitation meeting through the common Calendar Function.
- Creation of a Facilitation Meeting Event whenever a facilitation meeting has been conducted.

The Paternity Facilitation module generates ticklers, documents, and events as defined by the rules-based logic described within Section 2, Software Design of the Use Case document.

It creates, reads, updates, and/or deletes data from MTS tables as depicted in Section 3, Database of the Use Case Document.

The paternity facilitation design implementation uses the DocumentGenerationWebService to generate documents and the TicklerManagerWebService to create ticklers.
Conclusion:
The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case: Paternity Facilitation</td>
<td>MTS_UC_Paternity Facilitation.doc</td>
<td>Alfresco System -&gt; MTS -&gt; Design – Detail Design</td>
</tr>
</tbody>
</table>

5.3.4 MTS Use Case: Paternity Genetic Testing

Summary
This use case is comprised of processing that supports caseworkers with the scheduling of genetic tests and generation of documents to support genetic testing.

Approach
Caseworkers access UI panels to perform the activities of paternity genetic testing. The panels allow caseworkers to schedule genetic testing, record the results of testing, and maintain schedules of genetic testing laboratories.

The series of panels within paternity genetic testing allow the caseworkers to perform the following actions:

- Schedule Genetic Tests.
- Record Genetic Test Results.
- Record Genetic Test Schedule.
- Schedule Genetic Testing.

The module allows caseworkers to schedule a Genetic Test and generate an Appointment Letter for Genetic Test. If an appointment is missed, the module allows the worker to update the appointment and thus set an Appointment Missed Tickler through the Tickler Interface. The user interface for Genetic Testing Schedule uses the Calendar functionality in the system to schedule a Genetic Test.

Genetic Test Results
The module allows for capture of the following regarding genetic test results:

- Date the test was conducted.
- Time the test was conducted.
- Test Type.
- Docket number (for the case).
- Test Results (positive or negative).
- Date the test was ordered.
If results are negative, a Paternity not Established Tickler is set for the caseworker.

**Genetic Test Schedule**

Caseworkers view calendar of testing labs and schedule genetic testing. Data to capture and process lab availability data is modeled in the Calendar Use Case.

The Paternity Genetic Testing module generates ticklers, documents, and events as defined by the rules-based logic described within Section 2, Software Design of the Use Case document.

It creates, reads, updates, and/or deletes data from MTS tables as depicted in Section 3, Database of the Use Case Document.

The paternity genetic testing design implementation uses the DocumentGenerationWebService to generate documents and the TicklerManagerWebService to create ticklers.

**Conclusion:**

The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
</table>

5.3.5 MTS Use Case: Paternity Stipulation

**Summary**

This use case is comprised of processing that supports caseworkers with generation of documents to support voluntary acknowledgment of paternity and stipulations to paternity and child support.

**Approach**

Caseworkers access UI panels to perform the activities of paternity stipulation. The panels allow caseworkers to select next appropriate stipulation action and use the common Document Generation Interface to generate facilitation documents and common Tickler Interface to generate the related ticklers.

This module supports the caseworker with the following:

- **Generation of the Voluntary Acknowledgement of Paternity – NCP (#533).** If paternity is acknowledged, the module creates a Voluntary Acknowledgement Event.
- **Generation of a Hearing Waiver – Paternity (#344), a Stipulation to Paternity (#285), and a Paternity Order (#293) for judicial review (if the NCP wishes to stipulate to paternity, but has not requested a hearing or a genetic test).**
- **Generation of a Hearing Waiver – Paternity (#344), a Stipulation to Paternity and Child Support (#495), and a Paternity and Child Support Order (#498) for judicial review (if the NCP wishes to stipulate to paternity and child support, but has not requested a hearing or a genetic test).**
- Generate a Hearing Waiver – Paternity (#344) and a Stipulation to Paternity with Genetic Test (#496) (if the NCP wishes to stipulate to paternity with a genetic test).
- Generate a Hearing Waiver – Paternity (#344) and a Stipulation to Paternity and Child Support with Genetic Test (#497) (if the NCP wishes to stipulate to paternity and child support with a genetic test).
- Generate a Paternity Order (#293) or a Paternity and Child Support Order (#498) as appropriate after genetic test results have been received.
- Create a Voluntary Acknowledgment Event when the documents related to voluntary acknowledgment are generated.
- Create a Stipulation Event whenever the documents related to stipulation are generated.

The paternity stipulation module generates ticklers, documents, and events as defined by the rules-based logic described within Section 2, Software Design of the Use Case document.

It module creates, reads, updates, and/or deletes data from MTS tables as depicted in Section 3, Database of the Use Case Document.

The paternity stipulation design implementation uses the DocumentGenerationWebService to generate documents and the TicklerManagerWebService to create ticklers.

**Conclusion:**

The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case: Paternity Stipulation</td>
<td>MTS_UC_Paternity Stipulation.doc</td>
<td>Alfresco System -&gt; MTS -&gt; Design – Detail Design</td>
</tr>
</tbody>
</table>
5.4 Order Establishment

The Order Establishment component will allow workers to manage court orders and court ordered obligations and produce associated documents. It will process information on existing and prior orders, link multiple orders, and automatically calculate support orders based on obligation guidelines. It will generate documents to facilitate meetings with participants in which they voluntarily acknowledge paternity and child support obligations, those among CP and NCP that stipulate child support. This system will monitor and identify cases for review and modification, generating a letter and tickler for cases that are due.

Exhibit 5-4 diagrams the use cases which comprise the functionality of Order Establishment, as required by the GSD and SRS documents:

- Order Data.
- Order Facilitation.
- Order Stipulation.
- Order Documents.
- Order Monitor.

Exhibit 5-4: Order Establishment Use Cases
5.4.1 MTS Use Case: Order Data Entry

Summary
This use case is comprised of processing to support caseworkers with search, entry and update of data related to court orders and obligations.

Approach
Caseworkers access UI panels to perform the activities of order entry and update. The panels allow caseworkers to view and manage orders and obligations at the case level.

The caseworker can select an order to view details or optionally enter a new order. This module supports maintenance of multiple orders on each case.

For each order selected, the system will display a list of obligations entered on a given order. The obligation detail allows entry of a payee, obligation amount, and obligation frequency, kinship care details, and arrears details.

The caseworker can view and manage orders on those cases where he/she is the assigned caseworker. The supervisor has the ability to manage orders on any case in the system.

The panels allow caseworkers to perform the following actions:
• View Orders.
• View Obligations for an Order.
• Enter a New Order.
• Enter a New Obligation.
• Guideline Details.

The Order Data Entry module generates events as defined by the rules-based logic described within Section 2, Software Design of the Use Case document.

It creates, reads, updates, and/or deletes data from MTS tables as depicted in Section 3, Database of the Use Case Document.

Conclusion:
The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case: Order Data Entry</td>
<td>MTS_UC_Order Data Entry.doc</td>
<td>Alfresco System -&gt; MTS -&gt; Design – Detail Design</td>
</tr>
</tbody>
</table>
5.4.2 MTS Use Case: Order Facilitation

**Summary**

This use case is comprised of processing that supports caseworkers with generation of documents to facilitate meetings with participants in which they will voluntarily acknowledge paternity and child support obligations.

**Approach**

Caseworkers access UI panels to perform the activities of order facilitation. The panels allow caseworkers to select next appropriate facilitation action and use the common Document Generation Interface to generate facilitation documents and common Tickler Interface to generate the related ticklers.

The caseworker can generate a Facilitation Letter – First (#298) or a Facilitation Letter – Second (#299). Producing these documents creates ticklers to monitor for results of the facilitation. The caseworker can also schedule a meeting with the parties within the case which results in generation of the Appointment Letter – Order Establishment (#493) or Appointment Letter – Modification (#529).

The Order Facilitation module generates ticklers, documents, and events as defined by the rules-based logic described within Section 2, Software Design of the Use Case document.

It creates, reads, updates, and/or deletes data from MTS tables as depicted in Section 3, Database of the Use Case Document.

The order facilitation design implementation uses the DocumentGenerationWebService to generate documents and the TicklerManagerWebService to create ticklers.

**Conclusion:**

The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case: Order Facilitation</td>
<td>MTS_UC_Order Facilitation.doc</td>
<td>Alfresco System -&gt; MTS -&gt; Design – Detail Design</td>
</tr>
</tbody>
</table>
5.4.3 MTS Use Case: Order Stipulation

Summary

This use case is comprised of processing that supports caseworkers with generation of documents and events for the CP and NCP to stipulate to child support. The system will support this action by using the common process for document generation.

Approach

Caseworkers access UI panels to perform the activities of order stipulation. The panels allow caseworkers to select next appropriate stipulation action and use the common Document Generation Interface to generate facilitation documents and common Tickler Interface to generate the related ticklers. Caseworkers will generate the following documents used in establishing a support obligation through judicial review:

- Stipulation to Child Support (#158).
- Hearing Waiver – Enforcement (#296).
- Petition for Support Order (#538).
- Child Support Order (#304).

The Order Stipulation module generates ticklers, documents, and events as defined by the rules-based logic described within Section 2, Software Design of the Use Case document.

It creates, reads, updates, and/or deletes data from MTS tables as depicted in Section 3, Database of the Use Case Document.

The order stipulation design implementation uses the DocumentGenerationWebService to generate documents and the TicklerManagerWebService to create ticklers.

Conclusion:

The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case: Order Stipulation</td>
<td>MTS_UC_Order Stipulation.doc</td>
<td>Alfresco System -&gt; MTS -&gt; Design – Detail Design</td>
</tr>
</tbody>
</table>
5.4.4 MTS Use Case: Order Documents

**Summary**

This use case is comprised of processing that supports caseworkers with generation of documents to support court proceedings for establishing a child support order. Workers will use functionality modeled in the Document Generation process to select and produce order documents.

**Approach**

Caseworkers access UI panels to perform the activities of manual order document generation. They use the common Document Generation Interface to generate documents to support court proceedings for establishing a child support order. This module supports the caseworker in generating the following order documents:

- Voluntary Service Acceptance Letter (#300).
- Hearing Notice – Order Establishment (#484).
- Petition for Support Order (#538).
- Petition to Determine Arrears and Judgment (#179).
- Order to Reduce Arrears to Judgment (#541).
- Child Support Order (#304).
- Motion for Continuance (#171).
- Order for Continuance (#544).
- Motion to Dismiss (#491).
- Order to Dismiss (#490).
- Summons – Order Establishment (#487).
- Affidavit of Mailing (#508).
- Affidavit of Direct Payments (#104).
- Financial Affidavit (#204).
- Joint Stipulation for Modification (#140).
- Letter Acknowledging Review Request (#325).
- Letter Advising of No Change (#330).
- Order for Modification (#329).
- Order to Exclude (#545).
- Review and Modification Request (#53).

The Order Documents module generates documents as defined by the rules-based logic described within Section 2, Software Design of the Use Case document.
It reads data from MTS tables as depicted in Section 3, Database of the Use Case Document. The order documents design implementation uses the DocumentGenerationWebService to generate documents.

**Conclusion:**

The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case: Order Documents</td>
<td>MTS_UC_Order Documents.doc</td>
<td>Alfresco System -&gt; MTS -&gt; Design – Detail Design</td>
</tr>
</tbody>
</table>
5.4.5 MTS Use Case: Order Monitor

Summary
The order monitor process is a scheduled background component that executes at pre-defined intervals. This component analyzes each case and generates the appropriate Case Function changes and any related ticklers and history events.

Approach
This process operates to analyze the actions and data of all qualifying cases to determine if a case function status change is warranted. The process performs the actions (in the following order):

- Sets the Case Function to “Locate” if an NCP has no verified address or no verified employer.
- Sets the Case Function to “Paternity” if an NCP has been located and has employment verified but does not have paternity established or presumed.
- Sets the Case Function to “Order Establishment” if an NCP has been located and has employment verified and has paternity established or presumed.
- Sets the Case Function to “Enforcement” if there is a change in Case Function.

If the case has been inactive and in an “Open” status for the number of days specified in configuration data, the order monitor will set a Case Inactivity Tickler.

This module generates ticklers, events, and case function status as defined by the rules-based logic described within the Use Case document.

The Order Monitor module creates, reads, and/or updates data from MTS tables as depicted in Section 3, Database of the Use Case Document.

It operates within the MTS as a scheduler component. The scheduler framework contains an XML configuration file that defines the schedule policies for the components to be scheduled.

Conclusion:
The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case: Order Monitor</td>
<td>MTS_UC_Order Monitor.doc</td>
<td>Alfresco System -&gt; MTS -&gt; Design – Detail Design</td>
</tr>
</tbody>
</table>
5.5 Enforcement

The Enforcement component serves to monitor compliance with support orders and support the initiation of enforcement actions based on tribal policy. It will support enforcement remedies including income withholding, contempt proceedings, license suspension, asset seizure, state referrals, and others as defined in the system configuration. It will monitor compliance with support orders, initiate enforcement actions, and generate documents. In cases where previous enforcement attempts have failed, it will re-initiate enforcement actions.

Exhibit 5-5 diagrams the use cases which comprise the functionality of Enforcement, as required by the GSD and SRS documents:

- Enforcement.
- Enforcement Monitor.

Exhibit 5-5: Enforcement Use Cases
5.5.1 MTS Use Case: Enforcement

Summary
This use case is comprised of processing that supports caseworkers in pursuing the enforcement remedies provided by tribal policies. This module allows the caseworker to initiate enforcement actions for each case based on the tribal arrears thresholds and other criteria as given in Configuration Data. The enforcement remedies include income withholding, contempt proceedings, license suspension, seizure of assets, referrals to states, and other tribe-specific remedies.

Approach
Caseworkers access UI panels to perform the activities of enforcement. The panels allow caseworkers to initiate enforcement actions for each case based on the tribal arrears thresholds and other criteria as given in Configuration Data. The panels support entry and recording of data related to enforcement actions. The module generates ticklers to alert the caseworker of enforcement actions and to change Case Function as appropriate in response to changes in case data.

This module supports the following enforcement actions:

- Income Withholding.
  - Foreign Income Withholding.
  - Voluntary Income Withholding.
  - Immediate Income Withholding.
  - Garnishment.
- Contempt.
- License Suspension.
- Seizure.
- Release.

The enforcement module generates ticklers, documents, and events as defined by the rules-based logic described within Section 2, Software Design of the Use Case document.

It creates, reads, updates, and/or deletes data from MTS tables as depicted in Section 3, Database of the Use Case Document.

Conclusion:
The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case: Enforcement</td>
<td>MTS_UC_Enforcement.doc</td>
<td>Alfresco System -&gt; MTS -&gt; Design – Detail Design</td>
</tr>
</tbody>
</table>
5.5.2 MTS Use Case: Enforcement Monitor

**Summary**

The enforcement monitor process is a scheduled background component that executes at pre-defined intervals. It will fetch the enforcement configuration information, analyze all cases, and determine what enforcement methods are appropriate.

**Approach**

This module will fetch the enforcement configuration information, analyze all cases, and calculate what enforcement methods are eligible on a case based on configuration data and NCP arrears.

It creates, reads, and/or deletes data from MTS tables as depicted in Section 3, Database of the Use Case Document.

The MTS application scheduler invokes the enforcement monitor process at pre-defined scheduled intervals. The scheduler framework contains an XML configuration file that defines the schedule policies for the components to be scheduled.

**Conclusion:**

The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case: Enforcement Monitor</td>
<td>MTS_UC_Enforcement_Monitor.doc</td>
<td>Alfresco System -&gt; MTS -&gt; Design – Detail Design</td>
</tr>
</tbody>
</table>
5.6 Financial Management

The financial management component supports the required financial transaction processing, including the entry of collections, distribution, and disbursement of payments toward support obligations. It will perform routine charging on qualifying accounts, as well as delinquency testing and generations of letters, billing notices, and quarterly statements. This subsystem incorporates these features, among others:

- Allows financial workers to post payments, create and maintain employer group models for posting large numbers of payments, and produce all accounting reports necessary to balance and generate the daily bank deposit slip.
- Automatically allocates and applies monies to debt accounts in an order as determined by federal allocation hierarchies and calculates and records reimbursed and un-reimbursed monies.
- Automatically disburses support payments to clients and third-party agencies, and provides a tracking mechanism for all checks disbursed through the MTS.
- Allows financial workers and supervisors to correct payments, adjust accounts for an individual case or participant, and/or perform other actions to correct account imbalances.
- Assists financial workers with reconciling depository accounts by tracking all daily deposits and disbursements processed in the system.
- Search for obligation accounts that are due, creating a charge transaction and account adjustment if a balance is due. Check obligation account for timely and complete collection, generating delinquency letters as required. Check obligation account issuance of billing notice, if applicable.
- Allows financial workers to view account summary information by case account or participant account, including collections history and disbursement history.

Exhibit 5-6 diagrams the use cases which comprise the functionality of Financial Management, as required by the GSD and SRS documents:

- Financial Payment Post.
- Financial Account Charging.
- Financial Delinquency Test Monitor.
- Monthly NCP Billing Notices.
- Financial Account.
- Financial Account Summary.
- Financial Collection Post.
- Financial Deposit Process.
- Financial Pass-through.
- Financial Distribution.
- Financial Disbursements.
- Bank File Extraction & Reconciliation Matching.
- Bank Reconciliation Statement.
- Monthly Check Escheatment.
- Financial Transaction Adjustments.
- Monthly Account History.
- Quarterly Payment History Notices.

Exhibit 5-6: Financial Management Use Cases
5.6.1 MTS Use Case: Financial Payment Posting

Summary

This use case is comprised of processing that supports financial workers with batch creation and payment entry. Workers can create a new batch, record payments, and reconcile and approve a batch in accordance with tribal policy and regulations.

Approach

Financial workers access UI panels to perform the activities of payment posting. The panels allow financial workers to create a new batch and record data for each payment (up to 25 payments per batch) within a batch. This module will support the following:

- Search for batches using multiple criteria.
- Allow payments to be applied to one or more cases.
- Save an incomplete batch of payments allowing for retrieval and completion at a later time.
- Enforce reconciliation of batch item counts and totals prior to approval.
- Allow approval of multiple batches by a supervisor.

Workers navigate through a series of panels during payment entry process for each batch. As batches are created (or selected) and payments entered, the system will maintain a reconciliation count of batches and batch amount totals. Reconciled batches will be assigned a “Reconciled” batch processing status and remain static until approved by a supervisor. Batches that do not balance will remain in an “Incomplete” status until the batch is manually reconciled.

Reconciliation ensures that balanced accounting entries are created from payment-related transactions when made available for collections posting. Accepted (reconciled) batches will be assigned an “Approved” status and remain in a wait state for further processing within the Collections Posting module.

The Financial Payment Posting module creates, reads, updates, and/or deletes data from MTS tables as depicted in Section 3, Database of the Use Case Document.

Conclusion:

The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case: Financial Payment Posting</td>
<td>MTS_UC_Financial Payment Posting.doc</td>
<td>Alfresco System -&gt; MTS -&gt; Design – Detail Design</td>
</tr>
</tbody>
</table>
5.6.2 MTS Use Case: Account Charging

Summary
The account charging process is a scheduled background component that executes at pre-defined intervals. It will operate as a daily automatic process (in sequence with other background processes) to search obligation accounts that are due based on charge date and apply the appropriate charge amount.

Approach
This module functions to monitor participant obligation accounts which have a monthly frequency amount due and produce a charge transaction, if due. A monthly charging cycle will exist for all cases, whereby this module calculates and tracks all ordered frequency amounts as monthly equivalents during Account Setup and charge accordingly via this function. The use of standardized monthly amounts is necessary in order to establish a regular charging cycle for financial monitoring and essential to the distribution process as it allows funds to be prorated equitably across multiple cases.

The account charging module operates as a pre-defined scheduled process to perform the following:

- Check the monthly amount due on accounts where the charge date equals the run date on the obligor’s account to determine if charging is required.
- Debit any balance amount due from obligor’s account into the appropriate debt accounts.
- Apply accrued late payment, interest charges, and fees, as applicable.
- Reset the monthly amount due date in obligor’s account.
- Create an “Account Debited” financial history event record.

The scheduler will administer initiation of this background function, ensuring that it runs at pre-defined times and in the correct sequence with dependent functions.

The account charging module generates events as defined by the rules-based logic described within Section 2, Software Design of the Use Case document.

It creates, reads, and updates data from MTS tables as depicted in Section 3, Database of the Use Case Document.

Conclusion:
The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case: Account Charging</td>
<td>MTS_UC_Account_Charging.doc</td>
<td>Alfresco System -&gt; MTS -&gt; Design – Detail Design</td>
</tr>
</tbody>
</table>
5.6.3 MTS Use Case: Financial Delinquency Monitor

The financial delinquency monitor process is a scheduled background component that executes at pre-defined intervals. It will operate as a daily automatic process (in sequence with other background processes) to check for collections against financial accounts with recurring current support and/or ordered arrears payments. It will also generate ticklers and notices on delinquent accounts.

**Approach**

This module determines if obligation accounts are due and acts on overdue accounts by generating a tickler and delinquency notice. It will execute as part of a scheduled daily process driven by case details captured during the account charging process. The module examines financial accounts (with an ordered amount) for timely and complete payments from the obligor. Accounts without collection activity for 35 days will produce a Notice of Delinquency (#203), set the Account Delinquent Indicator, and set a tickler for the assigned caseworker.

The financial delinquency testing monitor module generates ticklers, documents, and events as defined by the rules-based logic described within Section 2, Software Design of the Use Case document.

It creates, reads, and/or updates data from MTS tables as depicted in Section 3, Database of the Use Case Document.

The MTS application scheduler invokes the financial delinquency monitor process at pre-defined scheduled intervals. The scheduler framework contains an XML configuration file that defines the schedule policies for the components to be scheduled.

**Conclusion:**

The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
</table>
5.6.4 MTS Use Case: Monthly NCP Billing Notices

Summary
The NCP billing notices process is a scheduled background component that executes at pre-defined intervals. It will operate as a daily automatic process (in sequence with other background processes) to determine if monthly billing is due and summarize case obligations into a billing statement. Billing suppression will be the system default setting for income withholding cases. Authorized users will be permitted to suppress the monthly bill generation for any case.

Approach
This module produces a billing statement for each payor case, showing amounts as ordered, monthly amounts due, and due by date on each remittance coupon. The NCP billing notices process is performed as part of a scheduled daily process driven by case details captured during the account charging and delinquency testing processes.

NCP billing notices operates as a scheduled daily process to perform the following:

- Check the case billing indicator to determine if billing is required.
- Summarize case obligations.
- Summarize payments from the prior month.
- Provide remittance coupons to ensure accurate payment identification upon receipt.
- Produce the Billing Notice using the Document Generation function.

Qualifying cases are extracted and amounts summed from event transactions to derive amount totals. It extracts and merges report amounts, case and participant profile information into a formatted statement, based on tribal policy.

On-demand printing of notices is supported by functionality contained within the Document Print Management Use Case. Workers use functionality modeled in the common Document Generation process to view and print (image file) quarterly payment history notices.

The Financial Billing Notice module generates documents as defined by the rules-based logic described within Section 2, Software Design of the Use Case document.

It reads and updates data from MTS tables as depicted in Section 3, Database of the Use Case Document.

The MTS application scheduler invokes the NCP Billing Notice process at pre-defined scheduled intervals. The scheduler framework contains an XML configuration file that defines the schedule policies for the components to be scheduled.

Conclusion:
The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
</table>
5.6.5 MTS Use Case: Financial Account

Summary

This use case is comprised of processing that supports financial workers with managing financial accounts and debt accounts on a selected case. MTS workers use UI panels to select a case and view accounts. Workers may view and create particular types of debt accounts and put a financial account on hold or release an account that was on hold.

Approach

Financial workers access UI panels to perform the activities of financial accounts. The panels allow financial workers to create and maintain fee-type debt accounts, and maintain hold status information related to financial accounts. The financial account module automatically creates debt accounts based on obligations created from a court order (during Obligation Setup) or allocation processing conditions encountered (during Distribution). It will automatically create or update financial accounts based on the following conditions:

- A new case is created in the system.
- New participants are added to a case.

Workers may update specific information related to an existing debt account, create a new debt account (creating a debt account is limited to the “Fees” debt category only), or update hold status information related to an existing financial account.

Newly created and modified debt accounts will create an event transaction and require approval by financial supervisors (if required by configuration data). Once approved, debt accounts will become active records for monthly account charging serving as the official financial record for a case. All financial processing (distribution, charging, delinquency, monthly billing, among others) will be based upon the information tracked by these accounts. Debt accounts that are automatically created as part of Obligation Setup are not modifiable, as the obligation information is court ordered.

The financial accounts module creates, reads, updates, and/or deletes data from MTS tables as depicted in Section 3, Database of the Use Case Document.

Conclusion:

The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
</table>
5.6.6 MTS Use Case: Financial Account Summary

Summary

This use case is comprised of processing that supports financial workers in viewing financial account information related to a case/participant(s).

Approach

Financial workers access UI panels to perform the activities of viewing financial account summary data. The panels allow financial workers to review account summary information on cases or participants. The user interface will be comprised of a panel which displays descriptive account information related to a case/participant, as dictated by user-defined selection criteria. Once activated, the panel displays populates nine (9) scrollable data grids whose views can be dynamically altered based on selection of a new case or member.

This module displays the following case account information:

- Obligation Details.
- Current Support.
- Arrears.
- Judgments.
- Fees.
- Disbursements.
- Balances.
- Case Transactions.
- NCP Transactions.

The financial account summary module reads data from MTS tables as depicted in Section 3, Database of the Use Case Document.

Conclusion:

The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case: Financial Account Summary</td>
<td>MTS_UC_Financial_Account_Summary.doc</td>
<td>Alfresco System -&gt; MTS -&gt; Design – Detail Design</td>
</tr>
</tbody>
</table>
5.6.7 MTS Use Case: Collection Posting

Summary
This use case is comprised of processing that supports financial workers with recording all collection data associated with a payment item in accordance with tribal policy and regulations.

Approach
Financial workers access UI panels to perform the activities of collection posting. The panels allow financial workers to enter collection data related to each payment (batch item). This module will accommodate payments from individuals and/or employers, and will support the following:

- Searching for payment batches using multiple criteria.
- Allowing a collection to be applied to one or more cases.
- Setting the release date to seven (7) business days if the NSF Indicator is set for a payor.
- Allowing suspension of a collection for subsequent research and adjustment when a payor is unknown.
- Saving incomplete collection entries for retrieval and completion at a later time.
- Calculating collection totals and amounts.

Financial workers will navigate through a series of panels for posting each payment (batch item). The system will assign a status code to each collection item based on data entered by the financial worker. This status indicator will signify the readiness of a collection item for processing by the distribution processes. An NSF or collection item marked for suspense will not be processed by the distribution process. “Ready” collection items will be processed by automated financial distribution.

The financial collection posting module creates, reads, updates, and/or deletes data from MTS tables as depicted in Section 3, Database of the Use Case Document.

Conclusion:
The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case: Collection Post</td>
<td>MTS_UC_Collection Post.doc</td>
<td>Alfresco System -&gt; MTS -&gt; Design – Detail Design</td>
</tr>
</tbody>
</table>
5.6.8 MTS Use Case: Deposit Processing

Summary
This use case is comprised of processing that supports financial workers with creating deposits from approved batches.

Approach
Financial workers access UI panels to perform the activities of deposit processing. The panels allow financial workers to create deposit records from batches which have been approved in prior MTS processes (Payment Posting). This module will support the following:

- Search batches using multiple criteria.
- Provide bank depository information about where funds will be placed.
- Functional limit of one batch per deposit (up to 25 deposit items).
- Listing of all deposit batch items for review.
- Allow printing of a deposit slip containing all deposit items using MTS common print processes.

Financial workers will navigate through a series of panels for deposit of each batch. As batches are selected for deposit and payments entered, the system will maintain a deposit amount total.

On-demand viewing and printing of deposit slips will be supported by functionality modeled in the common Document Generation process.

The financial deposit processing module creates, reads, updates, and/or deletes data from MTS tables as depicted in Section 3, Database of the Use Case Document.

Conclusion:
The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case: Deposit Processing</td>
<td>MTS_UC_Deposit Processing.doc</td>
<td>Alfresco System -&gt; MTS -&gt; Design – Detail Design</td>
</tr>
</tbody>
</table>
5.6.9 MTS Use Case: Financial Pass-through

Summary

The financial pass-through process is a scheduled background component that executes at pre-defined intervals. It will operate as a monthly automatic process to distribute a designated portion of collections to qualifying current assistance accounts, when configured.

Approach

This module functions as a monthly process to apportion current support monies collected on behalf of a public assistance recipient, updating appropriate accounts equal to the qualifying pass-through amount. Pass-through distribution uses system data to determine whether a record qualifies for pass-through. This function checks for the following, among other criteria:

- The case is a Current Assistance case.
- A tribal TANF distribution has occurred during the past month.
- The pass-through balance for the CP is less than the threshold set by financial configuration.

Qualifying accounts are updated with the appropriate pass-through amount. The Pass-through Distribution module will use system data to determine whether a record qualifies for pass-through.

The system executes financial pass-through as a scheduled process. The scheduler will run this module on a monthly basis, administering its initiation among the other dependent monthly accounting processes such as Monthly Account History.

The financial pass-through module generates events as defined by the rules-based logic described within Section 2, Software Design of the Use Case document.

This module creates, reads, and updates data from MTS tables as depicted in Section 3, Database of the Use Case Document.

Conclusion:

The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
</table>
5.6.10 MTS Use Case: Financial Distribution

Summary
The financial distribution process is a scheduled background component that executes at pre-defined intervals. It will operate as a daily automatic process to distribute collections on a case among the appropriate current support, arrears, judgments, and fees accounts automatically based on an allocation by debt percentage.

Approach
This module functions to direct payment amounts automatically based on a debt percentage allocation into the respective current support, arrears, judgment, and fees accounts for each case to satisfy current and past due support obligations. It operates as a scheduled process to perform the following:

• Identify qualifying payments for distribution processing.
• Allocate payments by debt percentage, as required.
• Distribute payee funds across the respective obligations based on a pre-defined priority scheme (Distribution Hierarchy).
• Update financial accounts and debt accounts with resultant balances.
• Create financial event records to reflect the account transactions.

The Distribution module determines the proper course of payment distribution by examining each qualifying payment and associated case for certain conditions, and then distributes payee funds across the respective obligations based on a pre-defined priority scheme.

The financial distribution process will operate from:

• Distribution Producer.
• Distribution Consumer.

The Distribution Producer will be implemented as a scheduled job that will be invoked at pre-defined intervals by the scheduler. The producer is implemented to identify qualifying payments for distribution and posts JMS messages to a JMS Topic \textit{jms/DistributionTopic}. The Distribution Consumer is implemented as a Message Driven Bean listening on the JMS Topic \textit{jms/DistributionTopic} for incoming messages to be consumed. The consumer implements the process steps for: preparing debt accounts for distribution, allocating and distributing funds to Current Support, allocating and distributing funds to assigned arrears (debt accounts with a frequency amount), allocating and distributing funds to un-assigned arrears (debt account without a frequency amount), and distributing funds to suspense.

The Financial Distribution module generates events as defined by the rules-based logic described within Section 2, Software Design of the Use Case document.

This module creates, reads, and updates data from MTS tables as depicted in Section 3, Database of the Use Case Document.
## Conclusion:
The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
</table>
5.6.11 MTS Use Case: Financial Disbursements

**Summary**

The financial disbursement process is a scheduled background component that executes at predefined intervals. It will operate as a daily process to automatically disburse payee funds to the appropriate system-level account, create transaction records of fund transfers, and update tables to invoke the generation of a paper check.

**Approach**

This module functions as a scheduled process executed daily direct distributed payment amounts automatically from payee accounts to the disbursement system-level account. It updates the appropriate tables to support the generation of support checks, as configured.

The financial disbursement module operates as a scheduled process to perform the following:

- Identify qualifying payments for disbursement processing.
- Perform requisite validations and look-ups of the payee account.
- Create transactions for transfer of payee funds to the appropriate system-level account.
- Update payee and system-level accounts with resultant balances.
- Create a financial event record to reflect the disbursement transaction.

The financial disbursement module generates events as defined by the rules-based logic described within Section 2, Software Design of the Use Case document.

This module creates, reads, and updates data from MTS tables as depicted in Section 3, Database of the Use Case Document.

**Conclusion:**

The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case: Financial Disbursement</td>
<td>MTS_UC_Financial_Disbursement.doc</td>
<td>Alfresco System -&gt; MTS -&gt; Design – Detail Design</td>
</tr>
</tbody>
</table>
5.6.12 MTS Use Case: Bank File Extraction & Reconciliation Matching

**Summary**

This process performs a daily extraction of bank account transactions and matching against MTS tables as part of the account reconciliation process. It will retrieve all bank transactions created since the last extraction and place them into a MTS table format and perform a match to debit and credit transactions.

**Approach**

The Bank Statement Extraction and Reconciliation Matching process is a scheduled process that runs on a daily basis. It supports the MTS bank reconciliation process by providing a method for electronically extracting and routing account statement information using an open financial exchange (OFX) communication protocol.

This function will serve to coordinate HTTP messages between tribal agency and banking institution to obtain all new account transactions. These transactions are then formatted into MTS tables and reconciled against outstanding disbursement (payments) and deposit item (deposits) records.

The design for Bank Statement Extraction & Reconciliation Matching uses the OFX protocol standards (as described by OFX specification 2.1.1) and sign-on information to access financial transaction information and load MTS files for subsequent reconciliation processing. The process is modeled to fetch the bank transaction statement data over a secure HTTPS connection after sending the sign-on information to the designated financial institution.

The Bank Statement Extraction and Reconciliation process will use the standard Java connectivity API and the Open Source implementation of the XMLBeans technology to retrieve and process the bank transaction statement. The process will run as an automatic nightly process.

This function will run as a daily background process to perform the following:

1. Securely connect (HTTPS) and download the transactions from the Financial Institution based on a given start date and end date (Note: The start date will be the last Run date and the end date is the end of current day).
2. Parse the returned XML file, identify the transactions, and process them.
3. Create ticklers (Success or Failure).

Update the Reconciliation table with the processing results and generate a reconciliation report.

This module creates, reads, and updates data from MTS tables as depicted in Section 3, Database of the Use Case Document.
Conclusion:
The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
</table>
5.6.13 MTS Use Case: Bank Reconciliation Statement

**Summary**

This module allows authorized financial workers to access panels to view results from the daily automated reconciliation match, search transactions, resolve mismatches, create a bank reconciliation statement, and apply adjustment account balances. The panels will display all reconciled and outstanding deposit items and disbursement checks, including financial transactions created by the MTS application (system-level receipt transactions) or those derived from the bank (service charges, fees, interest).

**Approach**

The Bank Reconciliation module will allow financial workers to perform daily reconciliation of financial accounts transactions against a bank account statement (following completion of prior background processes). It is comprised of a series of user-interface panels that display reconciliation match results and current account balances. Financial workers can search and view transactions (disbursement checks and deposit receipts) and unmatched records to reconcile accounts.

This module will operate from the Bank Statement Extraction & Reconciliation Match process which imports, validates, and reconciles bank statement to all outstanding MTS financial transactions. Matched transactions are updated to become part of the current day statement balance calculations.

The File Match Summary panel displays statistics from the recent bank statement extract for both debit and credit accounts. Financial workers can scroll through the transaction detail and update unmatched transactions, designating them as either “Reconciled” or “Suspended”. Workers can produce a File Match Summary Statement.

The Bank Reconciliation Summary panel will summarize the balance of debit and credit transactions for the statement period, allowing the worker to apply any statement adjustments. Disbursement checks and debit transactions contained in `FIN_DISB_CHEK`, are summarized for reconciliation. Deposit receipts and credit transactions contained in `FIN_DEPOSIT_ITEM` are summarized for reconciliation.

This panel also includes list detail of transactions that comprise the summary amount for the following:

- Unmatched Transactions.
- System-level Receipt Transactions.
- Deposits in Transit.
- Outstanding Disbursements.
- Statement Adjustments.

Transactions not originating in the MTS financial system, such as service charges, interest, and fees, appear in the respective panel and are included in the account balance calculations.
This function produces a file match summary statement and bank reconciliation statement using functionality modeled in the common Document Generation process.

The bank reconciliation statement module generates documents as defined by the rules-based logic described within Section 2, Software Design of the Use Case document.

This module creates, reads, and updates data from MTS tables as depicted in Section 3, Database of the Use Case Document.

The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case: Bank Reconciliation</td>
<td>MTS_UC_Bank Reconciliation Statement</td>
<td>Alfresco System -&gt; MTS -&gt; Design – Detail Design</td>
</tr>
<tr>
<td>Statement</td>
<td>Statement.doc</td>
<td></td>
</tr>
</tbody>
</table>
5.6.14 MTS Use Case: Monthly Check Escheatment

**Summary**

Monthly escheatment will search and set qualifying disbursement transactions to escheatment status, based on system configuration parameters, as defined by tribal policy. The process will update financial account balances to reflect the reversal of disbursed funds in support of financial reporting.

**Approach**

This process performs a monthly background process to search for unclaimed disbursement checks that have been dormant for a period based on a system configured timeframe. Funds from escheatable checks will be reverted to the agency by updating the appropriate system and participant financial accounts. This function will run as a monthly background process to perform the following:

- Search for disbursement transactions with an outstanding status for a period greater than the dormancy holding period defined by system configuration.
- Update FIN_ACCOUNT, FIN_DEBTACCOUNT, and FIN_SYSTEM_ACCOUNT tables to reflect the reverted amount of the escheated check.

**Conclusion:**

The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case: Monthly Check</td>
<td>MTS_UC_Monthly Check Escheatment.doc</td>
<td>Alfresco System -&gt; MTS -&gt; Design – Detail Design</td>
</tr>
</tbody>
</table>
5.6.15 MTS Use Case: Transaction Adjustments

Summary

This use case is comprised of processing that supports financial workers with performing adjustments and change actions to resolve account imbalances.

Approach

Financial workers access UI panels to perform the activities of financial transaction adjustment. The panels support creation of adjustment transactions to accommodate for financial processing errors or other unexpected exceptions. Workers will select from a set of standard pre-defined adjustments or construct a non-standard adjustment transaction from user-selected accounts.

Supervisors can search, select, and approve adjustments, which are then applied to MTS tables. Once applied, automated process invocation for documents and ticklers will commence as dictated by each adjustment type.

The transaction adjustments module performs the following:

- Allows for standard (pre-defined) or non-standard (ad-hoc) change actions to be performed on one or more financial accounts.
- Allows supervisors to approve newly created transaction adjustments.
- Allows workers to record adjustment notes.
- Validates user-selected/entered accounts for the desired adjustment action.
- Invokes MTS common processes (based on adjustment type) to reissue checks and generate NSF letters and ticklers once approved.

Common MTS processes will be employed by this function for searching, worklist/tickler setting, and documentation generation, as dictated by the adjustment type. This document references MTS common processes as applicable within the transaction adjustments function.

The transaction adjustments module generates events as defined by the rules-based logic described within Section 2, Software Design of the Use Case document.

This module creates, reads, and updates data from MTS tables as depicted in Section 3, Database of the Use Case Document.

Conclusion:

The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case: Financial Transaction Adjustment</td>
<td>MTS_UC_Financial_Transaction_Adjustment.doc</td>
<td>Alfresco System -&gt; MTS -&gt; Design – Detail Design</td>
</tr>
</tbody>
</table>
5.6.16 MTS Use Case: Monthly Account History

Summary

The monthly account history process is a scheduled background component that executes at pre-defined intervals. It will record financial account balances and reset month-to-date balances in support of financial reporting.

Approach

This module will allow for a point-in-time recording of account and debt account balances, and reset month-to-date debit/credit balances for subsequent accrual. This function will run as a monthly background process to perform the following:

- Record financial account, debt account and system-level account balances (and other account-related data) each month.
- Reset month-to-date balances in FIN_ACCOUNT, FIN_DEBTACCOUNT and FIN_SYSTEM_ACCOUNT tables.

It creates, reads, and/or deletes data from MTS tables as depicted in Section 3, Database of the Use Case Document.

The MTS application scheduler invokes the monthly account history process at pre-defined scheduled intervals. The scheduler framework contains an XML configuration file that defines the schedule policies for the components to be scheduled.

Conclusion:

The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case: Monthly Account History</td>
<td>MTS_UC_Monthly Account History.doc</td>
<td>Alfresco System -&gt; MTS -&gt; Design – Detail Design</td>
</tr>
</tbody>
</table>
5.6.17 MTS Use Case: Quarterly Payment History Notices

**Summary**

The Quarterly Payment History Notices Process is a scheduled background component that executes at pre-defined intervals. It will operate as a scheduled process executed quarterly to determine if payment notices are due and produce a single child support payment history notices per case.

**Approach**

This module produces a quarterly statement to current and former assistance recipients who have assigned rights to support. It calculates the amount of payments collected and collections disbursed to the family for that quarterly period. This module performs the following:

- Determines if a quarterly payment history notice is due.
- Summarizes collections received from the payor and payments made to the payee for a single case.
- Produce a Payment History Notice containing:
  - Current obligation monthly amount for the payor.
  - Amount collected from the payor by month for the reporting quarter.
  - Amount paid to the payee by month for the reporting quarter.
  - Amount paid to the assignee by month for the reporting quarter.
  - Amount of back support owed by child.
  - Amount of back support owed for the case.

Qualifying cases are extracted and amounts summed from event transactions to derive totals for collections and payments by month. It will extract and merge report amounts and case and participant profile information into a formatted statement, based on tribal policy.

On-demand printing of notices is supported by functionality contained within the Document Print Management Use Case. Workers use functionality modeled in the common Document Generation process to view and print quarterly payment history notices.

This module reads or updates data from MTS tables as depicted in Section 3, Database of the Use Case Document.

The MTS application scheduler invokes the Quarterly Payment History Notice process at pre-defined scheduled intervals. The scheduler framework contains an XML configuration file that defines the schedule policies for the components to be scheduled.
Conclusion:
The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case: Quarterly Payment History Notice</td>
<td>MTS_UC_Quarterly_Payment_History_Notice.doc</td>
<td>Alfresco System -&gt; MTS -&gt; Design – Detail Design</td>
</tr>
</tbody>
</table>
5.7 Application Management

The application management component consists of the central framework that governs operation of the applications which comprise the MTS. This framework is tightly integrated to ensure efficient developmental and operational activities. The following subsection introduces the components involved in application management and provides additional information about caching, data validation, exception processing, and process scheduling.

Describes processes related to the retrieval, maintenance and control of the application software and data, as cited in Section I-4.7 of the SRS.

Exhibit 5-7 diagrams the use cases which comprise the functionality of Application Management, as required by the GSD and SRS documents:

- Application jBPM Framework.
- Application Cache.
- Application Data Validation.
- Application Exception Processing.
- Application Scheduled Process.
- UI Validators.
- Reporting Framework.

Exhibit 5-7: Application Management Use Cases
5.7.1 MTS Use Case: Application jBPM Framework

Summary

jBPM is a JBoss Enterprise Framework that will be used for business process management and process orchestration to create and automate MTS business processes and services. Developers will use jBPM to integrate of SOA services and automate workflows in MTS applications to reduce business process error, speed process execution, and enhance performance. The JBoss jBPM process designer will graphically represent the business process steps in order to facilitate a strong collaboration between business analyst and technical developer.

Because MTS business processes have well defined start and end points, and follow a set of pre-defined paths as part of their execution, such processes lend themselves to be modeled as a workflow which is a repeatable pattern of activity involving defined roles and resources. The jBPM workflow engine will deploy workflows and create instances of these processes that represent real life instances. Workflow modeling will reduce the need for programming logic, which would otherwise be needed to model, manage, and execute the business processes.

Approach

The jBPM framework supports creation and automation of MTS business processes and services that will reduce business process errors, speed-up process execution, and enhance overall system performance. It offers a programmatic structure for designing transactions and executing them using automated decisions, tasks, and sequence flows.

MTS will implement a process template (process definition) for each workflow identified in the MTS business environment. These process templates will be deployed on a J2EE compliant container (JBoss) and managed by a workflow engine (jBPM). MTS has two specific workflow needs:

- Create process instances and execute portions of it programmatically and leave the process instance in a pre-defined state. User actions control further execution of these process instances.
- Monitor all existing process instances in the system. This is a reporting requirement that will display the overall statues of all process instances (by process template).

jBPM provides APIs for creating, managing, and terminating process instances of the process definitions. These API is used to meet both the management and reporting requirements.

Both the management and reporting services will be modeled as a stateless session bean that provides transaction demarcation, pooling, and high availability and serves as a session façade (design pattern). The actual work to perform is delegated to worker classes that are aware of the underlying jBPM APIs.

Conclusion:

The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case: Application jBPM Framework</td>
<td>MTS_UC_Application jBPM Framework.doc</td>
<td>Alfresco System -&gt; MTS -&gt; Design – Detail Design</td>
</tr>
</tbody>
</table>
5.7.2 MTS Use Case: Application Cache

Summary

The MTS application will use cache processing to support static information and frequently-used and infrequently-updated data. The cache configuration will store data such as system start-up values, security profiles, and valid data values, among others, in order to improve processing efficiency of the application. MTS will employ a wrapper allowing the use of any commercial cache program such as the JBossCache, OSCache, or Oracle Java Caching products. The flexible design allows the cache to be created using any vendor’s caching product, without impact to the overall design of the MTS application.

This design will implement the JBossCache framework to support application functions used within the MTS application. Application initialization parameters, valid values, system messaging, application monitoring, security tables, and security profiles, among others, will be supported by application caching. While fully configurable, the cache can be easily modified to be replicated over a cluster to meet the demands of response times and application scalability.

Approach

Caching is a technique for improving the efficiency of an application. The concept of temporarily storing frequently-referenced data within the application will save the overhead of repeated trips into the database. The JBossCache framework will be used in MTS to facilitate caching of frequently accessed and near static data using the TreeCache framework. Application initialization parameters, valid values, system messaging, application monitoring, security tables, and security profiles will be supported by application caching.

The use case design will clearly define the process for caching static information and frequently used data within the MTS application. The cache configuration used in the MTS application will be configured as a singleton implementation with a method that requires instructions to start and stop the caching framework. The configureCache method within this singleton is invoked from a ServletListener class that implements the ServletContextListener with implementation of contextInitialized and contextDestroyed methods.

Conclusion:

The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case: Application Cache</td>
<td>MTS_UC_Application Cache.doc</td>
<td>Alfresco System -&gt; MTS -&gt; Design – Detail Design</td>
</tr>
</tbody>
</table>
5.7.3 MTS Use Case: Application Data Validation

Summary

The MTS application will store all valid field values in separate tables that will relate the data field and associated valid values for data validation. These values will be loaded into the system cache upon application startup. The system cache used for storing the validation data will not be updated directly. All modifications to the validation data must occur directly to data structures designed to support the validation process and will then be loaded to the cache during startup.

The validation process must be supported by a unique field name referenced by the associated valid values. The table structures will support the addition, modification and removal of field names as well as the addition, modification and removal of associated valid values and associated descriptions. For purposes of the online application and J2EE program execution, this will be considered static data.

The design implements the data validation process to support application functions used within the MTS application. Loading the values into the application caching program on startup time will increase performance of module load-up as well as panel load and flip performance. The common usage between the Flex 2 application and the J2EE application reduce maintenance and improves application validation consistency.

Approach

Data validation is a key component of all application design and functionality. Utilizing common data structures to populate a system cache will provide quick common, enterprise wide access to all data validation. The implementation of the validation process can be easily modified and easily scaled for cluster implementation.

As part of the application startup, the CacheInitializerServlet will load all column names and valid values into the application cache.

Once loaded the ValidValueWebService can be invoked and will return all valid values for the specified field name passed into the getValues method. The ValidValueVO will contain columnName, displayName and array of strings for the associated valid values.

Conclusion:

The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case: Application Data Validation</td>
<td>MTS_UC_Application Data Validation.doc</td>
<td>Alfresco System -&gt; MTS -&gt; Design – Detail Design</td>
</tr>
</tbody>
</table>
5.7.4 MTS Use Case: Application Exception Processing

**Summary**

The exception processing framework within the MTS will accommodate any condition that causes an exception. The framework will produce the application information, warnings, and errors encountered during a UI session that will be displayed as messages to the user. The Exception Handling & Logging framework will be based on Java Exception Hierarchy and Logging, which is based on the Log4J Logging API.

**Approach**

The exception processing framework governs the processes for error handling and modes of detection, isolation, and messaging that may occur during the operation of the application. Online, background, and scheduled processes will be capable of handling failures and warnings and will allow for the requisite error messaging, logging, reporting, and administrator notification as dictated by the particular error handling process.

System errors, program errors, navigation errors, and logging errors will be tracked and recorded by the application for system maintenance. Error processing will be evaluated differently based on the type of error and on the audience associated with the area in which the error occurs.

**Conclusion:**

The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case: Application Exception Processing</td>
<td>MTS_UC_Application Exception Processing.doc</td>
<td>Alfresco System -&gt; MTS -&gt; Design – Detail Design</td>
</tr>
</tbody>
</table>
5.7.5 MTS Use Case: Application Scheduled Process

**Summary**

The system will implement a pre-defined scheduled process for each background task that needs to fulfill a specific business function. These scheduled processes will then be fired by a J2EE compliant scheduler at pre-defined intervals. These processes are optimized to achieve maximum throughput for a particular task, usually prioritizing performance over flexibility.

Real-time asynchronous processing performs the necessary business functions and results are reported to an on-line user when applicable. This approach requires using JMS for messaging and notification.

The real-time asynchronous framework will provide a generalized mechanism for the following:

- Job Control (including start and stop).
- Parallel Processing.
- Fine-grained Transaction Control.
- Error Handling.
- Job Monitoring.

The scheduled task will be implemented as a stateless session bean. This session bean is business agnostic. It does not contain any business-specific logic. It provides transaction demarcation only. The actual work is delegated to a Worker class that is job specific. This Worker class performs the business logic by acting on the data described by the business requirement. Worker classes process the data that satisfies the business requirement and create JMS messages. These messages are consumed by Message Driven Beans (MDBs). Worker classes act as data readers and the MDB’s act as data modifiers.

**Approach**

The asynchronous scheduled automated process within MTS provides the framework necessary to support many requirements as identified in section II-1.9 Timeframes and II-2.6 Ticklers. In addition to these requirements, the daily, weekly, monthly, quarterly, semi-annual, and annual reporting requirements will also be supported.

The components below are required to implement an asynchronous scheduled process.

**Client**

The client requires an asynchronous processing facility from the business objects participating in a workflow. The client can be any type of application that has the capability to create and send JMS messages. The client can also be an EJB that needs to invoke another enterprise bean's business methods in an asynchronous manner. The client can use the services offered by the Service Locator pattern to look up or create EJBs, JMS services, and JMS objects as necessary. The client has to implement the Job interface as defined in the scheduler API.

**Request**

The Request is the message object created by the client and sent to the ServiceActivator via the message-oriented middleware. According to the JMS specification, the Request is an object that
implements the javax.jms.Message interface. The JMS API provides several message types, including TextMessages and ObjectMessages that can be used as request objects.

**ServiceActivator**

The ServiceActivator is the main class of the pattern. It implements the javax.jms.MessageListener interface, which is defined by the JMS specification. The ServiceActivator has an onMessage() method that is invoked when a new message arrives. The ServiceActivator parses (unmarshals) the message (request) to determine what needs to be done. The ServiceActivator may use the services offered by a Service Locator (see Service Locator) pattern to look up or create Business Service components such as EJBS.

**BusinessObject**

BusinessObject is the target object to which the client needs access in an asynchronous mode. The business object is a role fulfilled by either a session or an entity bean. It is also possible that the BusinessObject an external service instead of an entity bean. In most EJB applications, a Session Façade (see Session Façade pattern) may fulfill the role of the BusinessObject to provide the required services. In such cases, the Service Activator activates the workflow encapsulated in a Session Façade.

**Scheduling Service**

The scheduling service will implement a custom job interface to define job arguments to be used in the execute() method. These values will define the specific characteristics about the job execution including name, timeframes, and exceptions.

**Conclusion:**

The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case: Application</td>
<td>MTS_UC_Application Scheduled Process</td>
<td>Alfresco System -&gt; MTS -&gt; Design – Detail Design</td>
</tr>
<tr>
<td>Scheduled Process</td>
<td>doc</td>
<td></td>
</tr>
</tbody>
</table>
5.7.6 MTS Use Case: UI Validators

Summary
This use case describes how UI validators will operate within MTS applications. Validators ensure that user-entered data meets specific criteria before the application uses it. Extending the existing Flex validation to support expanded validation and standardized execution will assist developers with employing Validators within their user interface panels.

Approach
The document describes how Validators will be implemented in the MTS project and standards to be used by developers. It describes the following:

- Creating Multiple Fields Validators.
- Building a Custom Validator Class.

Validators are user-interface control components, defined within the Flex API, that govern client-side data validation. Validators examine input data from the user before transmitting it to the server.

Data validators are predefined for many common types of user-supplied data, such as a date, number, and currency values, ensuring that the values in the fields of a data model meet certain criteria.

Flex validators are used within the various MTS applications (.mxml) as follows:

- Trigger validation by using events.
- Trigger validation by using the default event.
- Trigger validation for data bindings.
- Trigger validation programmatically.
- Validate required fields.
- Enable and disable a validator.
- Use data binding to configure validators.

Conclusion:
The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case: UI Validators</td>
<td>MTS_UC_UI Validators.doc</td>
<td>Alfresco System -&gt; MTS -&gt; Design – Detail Design</td>
</tr>
</tbody>
</table>
5.7.7 MTS Use Case: Reporting

Summary
The use case defines the design for automated reporting and report archival and how the MTS application will furnish reliable and scalable summary and activity reports. This process focuses on reports triggered by background processes (that are scheduled and automated rather than those generated from a particular user action). Several of these scheduled/automated reports will be delivered as Portable Document Format (PDF) files to MTS users. Batch processes and asynchronous events (covered in detail in the “Scheduled Process Use Case”) will serve as triggers for such asynchronous reports.

Approach
During the document creation process the PDF file will be saved to the MTS database. The saved documents will be created with the responsible worker ID, case ID, and or participant ID to allow for future printing. The responsible worker will be able to search for documents using the case history and print the requested document.

The system will implement a pre-defined scheduled process for each background task that needs to serve a reporting requirement. These scheduled processes will be fired by a J2EE compliant scheduler at pre-defined intervals. These processes are optimized to achieve maximum throughput for a particular task, usually prioritizing performance over flexibility. JasperSoft’s JasperReports is the tool used to producing the range of outputs, including HTML, PDF, Excel XLS, CSV and XML file formats. It builds dynamic presentations from either static data, or data retrieved from a database table using an SQL query, Hibernate queries, and is designed to be integrated directly into Java/J2EE applications.

Conclusion:
The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case: Document Archival and Regeneration</td>
<td>MTS_UC_XXXXX.doc</td>
<td>Alfresco System -&gt; MTS -&gt; Design – Detail Design</td>
</tr>
</tbody>
</table>
5.8 Document Generation

The document generation function will support automated and manual document production, including the creation, review, distribution, and printing of all system-generated and user-defined documents. MTS functions will initiate documents as specified by systems rules and condition. Authorized workers will produce document manually from pre-defined document templates (selecting from predefined paragraphs, images, signatures, or free-form text) according to case specifics. Once selected, merged, and composed, the document can be previewed on-line or placed in a queue for print generation and archive.

Exhibit 5-8 diagrams the use cases which comprise the functionality of Document Generation, as required by the GSD and SRS documents:

- Document Data Merge.
- Document Mapping.
- Document Print Management.
- Administrative Templates.
- Disbursement Check Printing.
Figure 5.8.2 Document Generation Architecture
5.8.1 MTS Use Case: Document Data Merge

**Summary**

The use case will provide a detailed description of all the components required to assemble the dynamic and static data from the predefined template. This component is used during document generation.

**Approach**

Data merging component will parse the XML document template and extract all variable data and perform queries to populate the dynamic portions of the XML. The process continues until all variable data is replaced by data from the database and a final XML is prepared. This XML document is processed by the document generation process to generate the final output.

**Conclusion:**

The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
</table>

5.8.2 MTS Use Case: Document Mapping

**Summary**

The use case will provide a detailed description of all the components required to support the data mapping functionality to allow the assignment of data element to document templates.

**Approach**

Data mapping will be provided by assigning dynamic variables to one or many data elements within the application. The many data elements will be placed in groups and will be assignable within the template designer. These variables and data groups will be populated by predefined SQL queries.

**Conclusion:**

The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
</table>
5.8.3 MTS Use Case: Document Print Management

**Summary**
This use case defines the actual generation of the final document from the XML document once the user confirms the print button on the preview window.

**Approach**
The static data merged with the variable data from the merge component is processed by this component using the iText PDF generation framework. The generated document is created as a PDF file, while a copy of the same file is persisted into the database for reprinting at a later time. The print management process will support batch print processing and real-time print processing.

**Conclusion:**
The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case: Document Print Management</td>
<td>MTS_UC_Document Print Management.doc</td>
<td>Alfresco System -&gt; MTS -&gt; Design – Detail Design</td>
</tr>
</tbody>
</table>

5.8.4 MTS Use Case: Document Generation – Admin Templates

**Summary**
The use case will provide a detailed description of how to view, create, edit, and delete document templates.

**Approach**
Template designer panels will be used by administrator with sufficient privileges to create a new document template. For each template that is created, the user will be required to assign a functional area, unique document id, document name, header details, footer details, and the body section. The template designer also provides the ability to view, edit, and/or delete document templates. Each time a document is edited and saved, the system creates a minor version of the document template. This template becomes the current template for document generation process.

**Conclusion:**
The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case: Document Generation – Administrative Templates</td>
<td>MTS_UC_DocgenAdminTemplate.doc</td>
<td>Alfresco System -&gt; MTS -&gt; Design – Detail Design</td>
</tr>
</tbody>
</table>
5.8.5 MTS Use Case: Disbursement Check Printing

Summary
This use case operates to produce hardcopy checks from disbursement records for delivery to the payee. It operates as a daily scheduled process and uses a MICR-encoded printing device for producing the check.

Approach
Checks are produced when the daily scheduled process is invoked to retrieve all qualifying records from FIN_DISB_CHEK which then creates the disbursement transaction that moves money from the payee account to the automated disbursement account. The system uses the disbursement data to populate the items into an imageable MICR-encoded format, print the check, and update the status of the disbursement check record.

The disbursement check printing module generates documents as defined by the rules-based logic described within Section 2, Software Design of the Use Case document.

This module creates, reads, and updates data from MTS tables as depicted in Section 3, Database of the Use Case Document.

The MTS application scheduler invokes the Disbursement Check Printing process at pre-defined scheduled intervals. The scheduler framework contains an XML configuration file that defines the schedule policies for the components to be scheduled.

Conclusion:
The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case: Disbursement Check Printing</td>
<td>MTS_UC_Disbursement Check Printing.doc</td>
<td>Alfresco System -&gt; MTS -&gt; Design – Detail Design</td>
</tr>
</tbody>
</table>
5.9 Security and Administration

The security and administration component provides related business functions through the configuration and security administration module. The configuration modules allow administrators to customize a tribal installation to perform optional functions for system, financial and enforcement processing. The security and administration modules perform access control services and regulate MTS entry points.

Exhibit 5-9 diagrams the use cases which comprise the functionality of Security and Administration, as required by the GSD and SRS documents:

- Security Administration.
- User Access.
- User Calendar.
- User Tickler & Worklists.
- System Configuration.
- Security Privileges.

Exhibit 5-9: Security & Administration Use Cases
5.9.1 MTS Use Case: Security Administration

**Summary**

This use case is comprised of processing that supports MTS administrators workers with security administration functions and processes.

**Approach**

Security administration and application security will be divided into separate menu functions. The functions will isolate security functions related to user maintenance, role/profile maintenance, and menu navigation maintenance. The goal for security will be to isolate access to screen navigation with menu restrictions based on role. Then assign read/write attributes for presentation pages based on the role provided, thus restricting the user to view/update of existing pages based on the role of the current user. All of the capabilities will be maintained in the “security administration” panels accessed by MTS administrators.

The security administration module creates, reads, updates, and/or deletes data from MTS tables as depicted in Section 3, Database of the Use Case Document.

**Conclusion:**

The document is referenced is in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case: Security Administration</td>
<td>MTS_UC_Security Administration.doc</td>
<td>Alfresco System -&gt; MTS -&gt; Design – Detail Design</td>
</tr>
</tbody>
</table>
5.9.2 MTS Use Case: User Access

Summary

This use case describes the functionality associated with MTS user access and management. System access control will be provided by application access screens that will provide login, lockout and logoff functionality.

Approach

Access to MTS will be secured by processes that provide user authentication through a standardized login process, user lockout capabilities through user initiated process, and a standard logout process initiated by either the user or by a system-initiated logoff process.

This use case defines a user screen and will accept the entry of a user ID and password for an approved user. Upon recognition and verification of the user ID and password, the system will open an initial screen based on the user’s role.

A screen will be triggered by a user action selecting the lock screen button. The screen will display when the user has locked his screen. This screen will show no case or participant data. It will permit access to the system upon entry of the password of the user who locked the screen.

Inactivity or a logoff requested by the user will trigger a user logoff screen. This screen will show no case or participant data. The inactivity timer or direct user request will remove all session information and display the logoff screen.

Impact

Based on the user role security information the system navigation options will be modified to the appropriate initial screen.

Modifications to the underlying desktop will be required to support a button to lock the user session. Session information must be stored for an indefinite timeframe when the lock process has been selected.

System audit triggers must be created to track user access and login/logoff history. Also a modification to the desktop is required to support a button to execute the user initiated logoff process. Session information must be purged from the application and user state must be terminated.

The user access module creates, reads, updates, and/or deletes data from MTS tables as depicted in Section 3, Database of the Use Case Document.

Conclusion:

The document is referenced is in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
</table>
5.9.3  MTS Use Case: User Calendar

Summary
This use case is comprised of processing that supports the system, users, and external agencies with calendar functionality. It operates as a common process for viewing, tracking, and changing calendar events related to managing a child support case and its participants.

Approach
TCSE workers access UI panels to perform the activities of calendaring. The panels allow users to record calendar events, including multiple caseworkers for a given appointment. The module will automatically generate ticklers for caseworkers. Ticklers can be generated by other sources as well; therefore the calendar design will integrate seamlessly with the tickler and worklist functionalities. The UI design will be flexible and loosely coupled with the backend design in order to present the calendar information in different formats (such as daily, weekly, and monthly). The Calendar function will also initiate the generation of necessary documents and case events in addition to ticklers depending on the calendar event type.

This module generates documents as defined by the rules-based logic described within the Use Case document.

The user calendar module creates, reads, updates, and/or deletes data from MTS tables as depicted in Section 3, Database of the Use Case Document.

Conclusion:
The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case: User Calendar</td>
<td>MTS_UC_User Calendar.doc</td>
<td>Alfresco System -&gt; MTS -&gt; Design – Detail Design</td>
</tr>
</tbody>
</table>
5.9.4 MTS Use Case: User Tickler & Worklists

Summary

This use case is comprised of processing that supports the system, users, and external agencies with tickler functionality. It operates as a common process for generating ticklers either automatically in response to an action on or by the system or manually by a user.

Approach

This module will create ticklers and places them on the user’s worklist when pre-defined data conditions are met. The tickler contains a set date and a send date. The tickler appears on the users worklist screen when the send date equals the current date. The system will define business processes that execute as a scheduled process with an asynchronous dispatch mechanism to examine current data and create ticklers when pre-determined conditions are met. Ticklers can be created only on OPEN and SUSPENDED cases.

MTS users have the ability to create ticklers manually and enter the text, Set Date, End Date, and other supporting data if needed.

The tickler record includes the following details:

- Tickler Code.
- Name and Description.
- Set Date.
- Send Date.
- Created By (System/User).
- Sent to (Caseworker).
- Case Number including Case Status and Function.
- Participants on Case.
- Caseworker ID.
- Business Process Instance ID (Used in cases where some business process is to be initiated which is in a wait state).
- Trigger Name (Used in cases where a pre-defined business process must be instantiated based on user action).
- Tickler Status (Open, Delete, Complete).

If a tickler is displayed on a worker’s worklist, the worker deletes the tickler, and the criteria are not satisfied for the tickler to be physically deleted, the system will automatically recreate the tickler and place it on the caseworker’s worklist on its next scheduled run.

Ticklers may also be generated and placed in the worker’s worklist by other system business functions, timeframe requirements, or other worker actions.

The user tickler and worklist module creates, reads, updates, and/or deletes data from MTS tables as depicted in Section 3, Database of the Use Case Document.
Conclusion:

The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case: User Tickler and Worklist</td>
<td>MTS_UC_User Tickler and Worklist.doc</td>
<td>Alfresco System -&gt; MTS -&gt; Design – Detail Design</td>
</tr>
</tbody>
</table>
5.9.5 MTS Use Case: System Configuration

**Summary**

The system configuration function involves the initial configuration of MTS processing options and allows user view and update. The initial installation process will consist of an automated configuration load of application, documents, ticklers, and system-level financial accounts. System Administrators will access configuration panels through the System Administration function to maintain the system configuration parameters. System Administrators will use this function to define initial control parameters for processing, including Application, Documents, and Tickler configurations.

Related to this function is the automated initialization and loading of system-level accounts that will be used by financial sub-systems.

**Approach**

Automated loaded processes will initialize and populate the respective tables allowing administrators to configure options for processing during initial installation. Certain fields will also be available to the system administrator during normal production operations. This module will allow configuration of the following components:

- Tribal Agency Information.
- Tribal Policies.
- Timeframes.
- Enforcement.
- Financials.
- Printers.
- Documents.
- Ticklers.
- System-level financial accounts.

As part of system setup, the CNFG_APPLICATION, CNFG_DOCUMENT, and CNFG_TICKLER tables will be created and loaded with default values using a SQL file script. Subsequently defined system configurations will be available in the backups of MYSQL tables. Tabbed panels will be accessed by the MTS Administrator to manage the Application, Documents and Ticklers panels are defined in Section 2.4.2.2, Flex MXML.

As part of system setup, System-Level Accounts will be created and loaded with default values using a SQL file script. This script will be run as part of system installation to populate the financial system-level account table (FIN_SYSTEM_ACCOUNT) for the respective accounts, each initialized with a zero balance.

The system configuration function creates, reads, and updates data from MTS tables as depicted in Section 3, Database of the Use Case Document.
Conclusion:
The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case: System Configuration</td>
<td>MTS_UC_System Configuration.doc</td>
<td>Alfresco System -&gt; MTS -&gt; Design – Detail Design</td>
</tr>
</tbody>
</table>
5.9.6 MTS Use Case: Security Privileges

Summary

The system will be configured to define role-based privilege criteria for each role. User interface panels will employ these criteria to determine if a specific role has “Read” capability or “Read/Write” capability on a given panel.

Approach

The design uses a relational table to define the role-specific privilege criteria. The MTS application loads privileges for each role during application startup into the application cache.

The user interface component of the security privileges will allow authorized system administrators to query the cache to get privilege data for the user during the authentication process. Privilege data is then used to dynamically enable or disable buttons on panels depending on whether the user has “Read” or “Read/Write” privilege for the current panel within a module.

As part of system setup, The CMMN_PRIVILEGE and CMMN_ROLE tables will be created and loaded with values using a SQL file script. As part of application startup, the Cache Initializer Servlet is responsible for loading the privilege data for each role into MTS Data Cache which is implemented using JBoss Cache.

The security privileges function creates, reads, and updates data from MTS tables as depicted in Section 3, Database of the Use Case Document.

Conclusion:

The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case: Security Privledges</td>
<td>MTS_UC_Security Privledges.doc</td>
<td>Alfresco System -&gt; MTS -&gt; Design – Detail Design</td>
</tr>
</tbody>
</table>
5.10 Common Functions

This section presents common functions that are shared among the various MTS applications. These applications interact with selected functions, providing common services to complete a routine business function.

Exhibit 5-10 diagrams the use cases which comprise common functions, as required by the GSD and SRS documents:

- Common Monitor.
- Address Management.
- Member Match.
- Create Financial Accounts.
- Create Debt Accounts.

Exhibit 5-10: Common Function Use Cases
5.10.1 MTS Use Case: Common Monitor

Summary

The common monitor function supports processing of ticklers, events, and documents for all cases among the various MTS program functions. This function is comprised of a series of scheduled background components that execute separately at pre-defined intervals to examine case, participant, and event data and generate tickler(s), event(s), and/or document(s) as required by the individual module.

Approach

The common monitor is comprised of a series of scheduled background processes that execute at pre-defined intervals. The MTS application scheduler invokes each monitoring job to examine case/participant data and determine whether particular conditions for a tickler, event, and/or document generation have been met. Business data conditions will spawn generate tickler(s), event(s), and/or document(s) as defined within each common monitor module listed below.

The common monitor consists of the following modules:

- Intake Document Monitor.
- Intake Alert Monitor.
- Locate Letter Monitor.
- Locate Time Monitor.
- Paternity Letter Monitor.
- Establishment Letter Monitor.
- Order Review Monitor.
- Order Alert Monitor.
- Enforcement Alert Monitor.
- Service of Process Monitor.
- Case Function Monitor.
- Financial Tickler Monitor.
- Management Alert Monitor.
- Emancipation Monitor.
- Case Closure Monitor.

The MTS application scheduler invokes the above functions at pre-defined scheduled intervals. The scheduler framework contains an XML configuration file that defines the schedule policies for the components to be scheduled.

Worklist Flex panels will be implemented to process and display all qualifying tickers in the caseworker’s worklist on the day specified by the send date. The Worklist panels will gather ticklers from the CMMN_TICKLER table by selecting all records where the Send Date equals the current date.
The common monitor generates ticklers, documents, and events as defined by the rules-based logic described within Section 2, Software Design of the Use Case document.

This module creates, reads, and updates data from MTS tables as depicted in Section 3, Database of the Use Case Document.

### 5.10.1.1 MTS Use Case: Intake Document Monitor

The intake document monitor is a scheduled background component that executes at pre-defined intervals. The MTS application scheduler invokes this module to examine data and determine whether particular conditions for tickler, event, and/or document generation have been met. This module will analyze each case to perform the following:

- Set and send a Notice of Non-Compliance Tickler if no response has been received to a Non-Cooperation Warning Letter (#14) within the number of calendar days specified by the caseworker.

- Generate a Notice of Cooperative Status (#15) document if a Notice of Non-Compliance Tickler is sent and the system is configured for automatic generation of the document.

### 5.10.1.2 MTS Use Case: Intake Alert Monitor

The intake alert monitor is a scheduled background component that executes at pre-defined intervals. The MTS application scheduler invokes this module to examine data and determine whether particular conditions for a tickler, event, and/or document generation have been met. This module will analyze each case to perform the following:

- Set and send a Birthday Tickler when a Child Date of Birth equals the current date.

- Set and send an Information Request Tickler when a Tribal Enrollment Verification Form (#73) is generated.

- If a Child Support Enforcement Transmittal #1 – Tribal (#261) or a Child Support Enforcement Transmittal #1 (#248) is generated, the system will generate a Foreign Referral Event.

- If the CP has an open case in a foreign jurisdiction, the system will generate Notice to Close Foreign Case (#346) document.

- Set and send a Foreign Case Closure Tickler and generate a Foreign Case Notice Event when a Notice to Close Foreign Case (#346) document is generated.

- Set and send a Family Violence Tickler when the Family Violence Code value is set to Temporary.

- Set and send a Case Suspended Tickler when the Case Status is set to Suspended.

- Send the following ticklers set by the caseworker when the Send Date equals the current date:
  - Foreign Referral Tickler
Worker Generated Tickler

5.10.1.3 MTS Use Case: Locate Letter Monitor
The locate letter monitor is a scheduled background component that executes at pre-defined intervals. The MTS application scheduler invokes this module to examine data and determine whether particular conditions for tickler, event, and/or document generation have been met. This module will analyze each case to perform the following:

- If no response has been received to a Locate Letter (Document #8, #9, #272, and #275) with the number of calendar days specified by the caseworker when setting the tickler, the system will automatically set and send a Locate Letter Tickler.
- If no response has been received to a Postmaster Letter (#468) with the number of calendar days specified by the caseworker when setting the tickler, the system will automatically set and send a Postmaster Letter Tickler.
- If no response has been received to an Employment Verification Letter (#6) with the number of calendar days specified by the caseworker when setting the tickler, the system will automatically set and send an Employment Verification Letter Tickler.

5.10.1.4 MTS Use Case: Locate Time Monitor
The locate time monitor is a scheduled background component that executes at pre-defined intervals. The MTS application scheduler invokes this module to examine data and determine whether particular conditions for tickler, event, and/or document generation have been met. This module will analyze each case to perform the following:

- Set and send a No Address Tickler if a case has had no NCP address for the number of calendar days specified by system configuration.
- Set and send a No Verified Address Tickler if a case has had an unverified NCP address for the number of calendar days specified by system configuration.
- Set and send a No Verified Employer Tickler if a NCP is employable and has an employer and the employer has not been verified for the number of calendar days specified by system configuration.
- Set and send a No Employer Tickler if an NCP is Employable and has had no employer for the number of calendar days specified by system configuration.

5.10.1.5 MTS Use Case: Paternity Letter Monitor
The paternity letter monitor is a scheduled background component that executes at pre-defined intervals. The MTS application scheduler invokes this module to examine data and determine whether particular conditions for tickler, event, and/or document generation have been met. This module will analyze each case to perform the following:
• If no response has been received to a Paternity Interview Letter – First (#276) within the number of calendar days specified by the caseworker, the system will automatically set and send a Paternity Interview One Tickler.

• If no response has been received to a Paternity Interview Letter – Second (#278) within the number of calendar days specified by the caseworker, the system will automatically set and send a Paternity Interview Two Tickler.

• If no response has been received to a Voluntary Acknowledgment of Paternity Letter – First (#287) with the number of calendar days specified by the caseworker, the system will automatically set and send a VAP Letter One Tickler.

• If no response has been received to a Voluntary Acknowledgment of Paternity Letter – Second (#513) with the number of calendar days specified by the caseworker, the system will automatically set and send a VAP Letter Two Tickler.

5.10.1.6 MTS Use Case: Establishment Letter Monitor

The establishment letter monitor is a scheduled background component that executes at pre-defined intervals. The MTS application scheduler invokes this module to examine data and determine whether particular conditions for tickler, event, and/or document generation have been met. This module will analyze each case to perform the following:

• If no response has been received to a Facilitation Letter – First (#298) within the number of calendar days specified by the caseworker, the system will automatically set and send a Facilitation One Tickler.

• If no response has been received to a Facilitation Letter – Second (#299) within the number of calendar days specified by the caseworker, the system will automatically set and send a Facilitation Two Tickler.

• If no response has been received to a Voluntary Service Acceptance Letter (#300) with the number of calendar days specified by the caseworker, the system will set and send a Voluntary Service Acceptance Letter Tickler.

5.10.1.7 MTS Use Case: Order Review Monitor

The order review monitor is a scheduled background component that executes at pre-defined intervals. The MTS application scheduler invokes this module to examine data and determine whether particular conditions for tickler, event, and/or document generation have been met. This module will analyze each case to perform the following:

• If a case has an established order and it has been number of calendar days determined by the tribe since the last review, the system generates a Letter Advising of Review Eligibility (#480) and generates a Review Notice Tickler to the caseworker.

• If the system has generated the Letter Advising of Review Eligibility (#480) and it has been number of days defined by the tribal policy with no response from either party, the system will reset the date for review eligibility.
5.10.1.8 MTS Use Case: Order Alert Monitor

The Order Alert monitor is a scheduled background component that executes at pre-defined intervals. The MTS application scheduler invokes this module to examine data and determine whether particular conditions for tickler, event, and/or document generation have been met. This module will analyze each case to perform the following:

- Generate an Order Change Tickler when an obligation is due to change.
- Set and send a Review Request Tickler when a Letter Acknowledging Review Request (#325) document is generated.
- Send a Pending Action Tickler (set by the caseworker) when the Send Date equals the current date.
- Send a Review Notice Tickler when the Send Date equals the current date.

5.10.1.9 MTS Use Case: Enforcement Alert Monitor

The enforcement alert monitor is a scheduled background component that executes at pre-defined intervals. The MTS application scheduler invokes this module to examine data and determine whether particular conditions for tickler, event, and/or document generation have been met. This module will analyze each case to perform the following:

- Set and send a Voluntary IWO Tickler when a Voluntary Wage Withholding Agreement (#471) and Notice of Voluntary Wage Withholding (#472) document is generated by the system.
- Set and send a Remedial Contempt Tickler when a Remedial Contempt Order (#319) document is generated.
- Set and send an IWO Tickler when an Income Withholding Order – Federal (#505) and an Income Withholding Letter to Employer (#499) document for the NCP and employer is generated by the end-user.
- Set and send a Failure to Withhold Notice Tickler when a Notice of Failure to Withhold (#311) document is generated by the end-user.
- Set and send a License Suspension Tickler when an Order to Show Cause (#318) or an Order to Suspend Tribal License (#474) has been generated.
- Set and send a Tribal Warning Letter Tickler when a License Suspension NCP Tribal Warning Letter (#55) document is generated by the end-user.
- Set and send a Seizure Warning Letter Tickler if a Seizure Warning Letter (#320) document has been generated.
- Set and send a State Notice Tickler if a Child Support Enforcement Transmittal #1 (#248) document has been generated to request a state to suspend a state license.
5.10.1.10 MTS Use Case: Service of Process Monitor

The service of process monitor is a scheduled background component that executes at pre-defined intervals. The MTS application scheduler invokes this module to examine data and determine whether particular conditions for tickler, event, and/or document generation have been met. This module will analyze each case to perform the following:

- Generate a Bad Address Tickler if SOP was unsuccessful due to a bad address.
- Set and send a Reschedule Hearing Tickler when a Hearing Notice - Cancellation (#523) document is generated.

5.10.1.11 MTS Use Case: Case Function Monitor

The case function monitor is a scheduled background component that executes at pre-defined intervals. The MTS application scheduler invokes this module to examine data and determine whether particular conditions for tickler, event, and/or document generation have been met. This module will analyze each case to perform the following:

- If a case has “Open” status and has been inactive for the number of calendar days determined by the tribe, the system will set and send a Case Inactivity Tickler for the caseworker’s supervisor.
- If a case has “Open” status and the NCP does not have a verified address the system will set the Case Function to “Locate”.
- If a case has “Open” status the system will set the Case Function to “Locate” if the NCP:
  - Has a verified address.
  - Is employable, but does not have a verified employer.
- If a case has “Open” status, the system will set the Case Function to “Order Establishment” if the NCP:
  - Has a verified address.
  - Is either not employable or has a verified employer.
  - There is no valid child support order.
  - Paternity is either established or presumed.
- If a case has “Open” status, the system will set the Case Function to “Paternity” if the NCP:
  - Has a verified address.
  - Is either not employable or has a verified employer.
  - There is no valid child support order.
  - Paternity is neither established nor presumed.
- If a case has “Open” status, the system will set the Case Function to “Enforcement” if the NCP:
o Has a verified address.
o Is either not employable or has a verified employer and there is a valid child support order and payments are not delinquent.

• If a case has “Open” status, the system will set the Case Function to “Delinquent” if the NCP:
o Has a verified address.
o Is either not employable or has a verified employer.
o There is a valid child support order and payments are delinquent.

• If the system has changed the Case Function, the system will generate a Case Status Change Event.

• If the Case Function has transitioned to “Enforcement”, the system will set and send a Case in Enforcement Tickler.

• If the Case Function has transitioned to “Order Establishment”, the system will set and send a Case in Order Establishment Tickler.

5.10.1.12 MTS Use Case: Financial Tickler Monitor

The financial tickler monitor is a scheduled background component that executes at pre-defined intervals. The MTS application scheduler invokes this module to examine data and determine whether particular conditions for tickler, event, and/or document generation have been met. This module will analyze each case to perform the following:

• Set and send an In-Kind Payment Received Tickler when an in-kind payment is recorded during Collection Posting.

• Set and send an NSF Adjustment Tickler when an NSF type adjustment is performed during Transaction Adjustments.

• Set and send a Check Escheated Tickler when Disbursement Check Status changes from "Outstanding" to "Escheated".

5.10.1.13 MTS Use Case: Management Alert Monitor

The management alert monitor is a scheduled background component that executes at pre-defined intervals. The MTS application scheduler invokes this module to examine data and determine whether particular conditions for tickler, event, and/or document generation have been met. This module will analyze each case to perform the following:

• Set and send a SOP Initiated Tickler when service of process is initiated.

• Set and send an Appointment Letter Tickler when a Appointment Letter - Intake (#252) document is generated.
- Set and send a Consolidation Notice Tickler when a Letter Advising of Order to Consolidate (#476) document or Notice/Motion for Consolidation (#331) document is generated.
- Set and send a Release of Information Tickler when a Release of Information Form (#71) document is generated.
- Set and send a Transfer Tickler when a Letter Advising of Transfer of Jurisdiction (#477) document is generated.
- Set and send a Hearing Scheduled Tickler when a hearing is scheduled.

5.10.1.14 MTS Use Case: Emancipation Monitor

Emancipation monitor is a scheduled background component that executes at pre-defined intervals. The MTS application scheduler invokes this module to examine data and determine whether particular conditions for tickler, event, and/or document generation have been met. This module will analyze each case to perform the following:

- Set and send an Emancipation Pending Tickler, three months before the date of emancipation of any child on a case.

5.10.1.15 MTS Use Case: Case Closure Monitor

The case closure monitor is a scheduled background component that executes at pre-defined intervals. The MTS application scheduler invokes this module to examine data and determine whether particular conditions for tickler, event, and/or document generation have been met. This module will analyze each case to perform the following:

- Set and send a Case Closure One Tickler if a Case Closure Notice – First (#72) document is generated.
- If a case has an active Case Closure One Tickler and it has been more than the specified time for the response set by the tribal policy, the system will generate the Case Closure Notice – Second (#518) and set and send a Case Closure Two Tickler.
- Set and send a Case Closure Notice Tickler when a Case Closure Request (#103) document or Case Closure Notice – First (#72) notice is generated.
- Set a Case Kept Open Event when a Case Closure Notice – Denied (#519) document is generated.
- If the caseworker closes the case, the system will create a Case Closure Notice – Closed (#520) for both parties, generate a Close Case Event, set and send a Closure Review Tickler to the case supervisor for review, set and send a Supervisory Review Tickler when a Case Closure Notice – Closed (#520) document is generated.
- If a response is not received within the timeframe, the system will set and send a Close Case Tickler to the caseworker to close the specified case.
Detailed System Design

- If all children on a case have been emancipated, the system will send a Case Closure – Emancipation Tickler.
- If paternity for the NCP on a case has not been established for one year, the system will set and send a Case Closure – Paternity Tickler.
- If the location for the CP on a case has been unknown for one year, the system will set and send a Case Closure – CP Lost Tickler.
- If the location of an NCP on a case has been unknown for three years, the system will set and send a Case Closure – NCP Lost Tickler.
- If a case is arrears only and arrears are $0, the system will set and send a Case Closure – Arrears Only Tickler.
- If Good Cause is set for a Case, the system will set and send a Case Closure – Good Cause Tickler.
- If the child support order for a case has expired, the system will set and send a Case Closure – Order Expired Tickler.

5.10.2 MTS Use Case: Member Match

**Summary**

This use case is comprised of processing that supports caseworkers with searching for participants and his or her related cases using First Name, Last Name, Date of Birth (DOB), Sex, and/or Social Security number (SSN). It will optionally perform a phonetic search.

**Approach**

Member match is a generic process that is part of the Common Popup Screen Use Case. It provides a list of participants based on pre-defined filter criteria. The function can be invoked from any functional area to search and select a participant and/or view cases for a participant. Selecting a participant from the list returns the participant object to the caller.

Financial workers access UI panels to perform the activities of member match. The panels allow caseworkers to enter the participant’s First Name, Last Name, DOB, Sex, and SSN (or any combination) and click “Search” to search the database to find matching participants. The worker also has the option to perform a Soundex search by selecting the “Soundex Search” checkbox on the search panel.

The search function displays list of participants that match the input criteria entered. The caseworker can select a participant from the list to view participant details by selecting the “Member Details” button, view cases for the selected participant by selecting the “Member Cases” button, and/or select the participant object by selecting the “Select Member” button.

The common member match screen can be invoked from any component/module within any functional area by invoking the display popup command and setting the current state to Member Search.

This module reads updates data from MTS tables as depicted in Section 3, Database of the Use Case Document.
Conclusion:
The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case: Member Match</td>
<td>MTS_UC_Member_Match.doc</td>
<td>Alfresco System -&gt; MTS -&gt; Design – Detail Design</td>
</tr>
</tbody>
</table>
5.10.3 MTS Use Case: Create Financial Accounts

Summary

Financial account creation is a common function which is invoked by a calling process to construct financial accounts for a case and/or participant, as required.

Approach

This module will be invoked from a MTS program function to construct and insert a financial account record into the FIN_ACCOUNT table. The invoking process will specify parameters that define the financial account to be created. Once created and inserted into the table, this process will then create and insert a corresponding event record into the FIN_EVENT table and return a status message to the caller, with an indicator when an error is encountered.

The financial account creation function will be implemented as a Java Singleton class allowing other functional areas within the system to create debt accounts. When invoked, the financial account creation function will produce a unique debt account (debt category & debt type combination) for a case.

An event record is created in FIN_EVENT for each successful instance of a new debt account inserted into FIN_ACCOUNT.

The financial account creation module generates documents as defined by the rules-based logic described within Section 2, Software Design of the Use Case document.

It creates, reads, updates, and/or deletes data from MTS tables as depicted in Section 3, Database of the Use Case Document.

Conclusion:

The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case: Create Financial Accounts</td>
<td>MTS_UC_Create Financial Accounts.doc</td>
<td>Alfresco System -&gt; MTS -&gt; Design – Detail Design</td>
</tr>
</tbody>
</table>
5.10.4 MTS Use Case: Create Debt Accounts

**Summary**

Debt account creation is a common function which is invoked by a calling process to construct debt accounts as required.

**Approach**

This module will be invoked from a MTS program function to construct and insert a debt account record into the FIN_DEBTACCOUNT table. The invoking process will specify parameters that define the debt account to be created. Once created and inserted into the table, this process will then create and insert a corresponding event record into the FIN_EVENT table and return a status message to the caller, with an indicator when an error is encountered.

The debt account creation function will be implemented as a Java Singleton class allowing other functional areas within the system to create debt accounts. When invoked, the debt account creation will produce a unique debt account (debt category & debt type combination) for a case.

An event record is created in FIN_EVENT for each successful instance of a new debt account inserted into FIN_DEBTACCOUNT.

The debt account creation module generates documents as defined by the rules-based logic described within Section 2, Software Design of the Use Case document.

It creates, reads, updates, and/or deletes data from MTS tables as depicted in Section 3, Database of the Use Case Document.

**Conclusion:**

The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case: Create Debt</td>
<td>MTS_UC_Create Debt Accounts</td>
<td>Alfresco System -&gt; MTS -&gt; Design</td>
</tr>
<tr>
<td>Accounts</td>
<td>Accounts.doc</td>
<td>– Detail Design</td>
</tr>
</tbody>
</table>
5.11 Third Party Data Management

This section presents functions related to the maintenance and configuration of third party information (e.g., employers, insurance companies, agencies, courts, and attorneys). This section supports the various MTS applications with third party information. Third party configuration relates to the initial load of third party data to the third party tables during system setup and configuration. Third party search describes a general search functionality of all third party entities. The additional third party use cases detail the maintenance of third party data, including searching against specific third party types, adding records third party records, and updating existing third party records.

Exhibit 5-11 diagrams the use cases which comprise the functionality of Third Party Data Management, as required by the GSD and SRS documents:

- Third Party Maintenance & Configuration.
- Third Party Search.
- Third Party Agency.
- Third Party Attorney.
- Third Party Correctional Facility.
- Third Party Court.
- Third Party Employer.
- Third Party Financial Institution.
- Third Party Genetic Testing Lab.
- Third Party Health Care Provider.
- Third Party Health Insurance Provider.
- Third Party Law Enforcement.
- Third Party Probation Parole Officer.
- Third Party Registered Agent.
- Third Party School.
- Third Party Tribal CSE Office.
Exhibit 5-11: Third Party Maintenance & Configuration Use Cases
5.11.1 MTS Use Case: Third Party Maintenance & Configuration

**Summary**

This use case involves the initial configuration of third party data and allows user view and update. The initial installation process will consist of an automated configuration load of third party information (e.g., employers, insurance companies, agencies, courts, attorneys, among others).

**Approach**

Automated loaded processes will initialize and populate the respective tables allowing administrators to maintain third party tables. As part of system setup, the respective tables will be created and loaded with default values using a SQL file script.

The third party maintenance & configuration function creates, reads, and, updates data from MTS tables as depicted in Section 3, Database of the Use Case Document.

**Conclusion:**

The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
</table>
5.11.2 MTS Use Case: Third Party Search

Summary
The third party search module allows users to search against all third party tables or select a specific table to search.

Approach
The third party search function allows users to search all third party tables. This function is accessed throughout the MTS application to search third party information by using specified search criteria (such as name and type) and viewing a list of matching third party records through the UI panels. Authorized users have the ability to select a record from the list and open the related detail popup for the third party record.

The third party search module displays a search form to allow the worker to enter selected criteria such as the third party name and optionally the third party type. Name is the only required entry. Selecting the “Search” Button will return a list of third party entities.

Conclusion:
The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case: Third Party Search</td>
<td>MTS_UC_Third_Party_Search.doc</td>
<td>Alfresco System -&gt; MTS -&gt; Design – Detail Design</td>
</tr>
</tbody>
</table>
5.11.3 MTS Use Case: Third Party Agency

**Summary**
This component operates with the third party search module to maintain information related to third party agencies.

**Approach**
MTS users can activate the third party agency popup from the third party search module. The popup allows the user to update information on third party agencies.

The third party agency module reads and/or updates data from MTS tables as depicted in Section 3, Database of the Use Case Document.

**Conclusion:**
The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case: Third Party Agency</td>
<td>MTS_UC_ Third Party Agency.doc</td>
<td>Alfresco System -&gt; MTS -&gt; Design – Detail Design</td>
</tr>
</tbody>
</table>
5.11.4 MTS Use Case: Third Party Attorney

**Summary**

This component operates with the Third Party Search module to maintain information related to third party attorneys.

**Approach**

MTS users can activate the Third Party Attorney popup from the third party search module. The popup allow user to update information on third party attorneys.

The Third Party Attorney module reads and/or updates data from MTS tables as depicted in Section 3, Database of the Use Case Document.

**Conclusion:**

The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case: Third Party</td>
<td>MTS_UC_ Third Party Attorney.doc</td>
<td>Alfresco System -&gt; MTS -&gt; Design – Detail Design</td>
</tr>
</tbody>
</table>
5.11.5 MTS Use Case: Third Party Correctional Facility

**Summary**
This component operates with the Third Party Search module to maintain information related to third party correctional facilities.

**Approach**
MTS users can activate the Third Party Correctional Facility popup from the Third Party Search module. The popup allow user to update information on third party correctional facilities.

The Third Party Correctional Facility module reads and/or updates data from MTS tables as depicted in Section 3, Database of the Use Case Document.

**Conclusion:**
The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case: Third Party Correctional Facility</td>
<td>MTS_UC_ Third Party Correctional Facility.doc</td>
<td>Alfresco System -&gt; MTS -&gt; Design – Detail Design</td>
</tr>
</tbody>
</table>
5.11.6 MTS Use Case: Third Party Court

**Summary**

This component operates with the Third Party Search module to maintain information related to third party courts.

**Approach**

MTS users can activate the Third Party Court popup from the third party search module. The popup allow user to update information on third party courts.

The third Party Court module reads and/or updates data from MTS tables as depicted in Section 3, Database of the Use Case Document.

**Conclusion:**

The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case: Third Party Court</td>
<td>MTS_UC_ Third Party Court.doc</td>
<td>Alfresco System -&gt; MTS -&gt; Design – Detail Design</td>
</tr>
</tbody>
</table>
5.11.7 MTS Use Case: Third Party Employer

**Summary**

This component operates with the Third Party Search module to maintain information related to third party employers.

**Approach**

MTS users can activate the Third Party Employer popup from the Third Party Search module. The popup allow user to update information on third party employers.

The Third Party Employer module reads and/or updates data from MTS tables as depicted in Section 3, Database of the Use Case Document.

**Conclusion:**

The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case: Third Party Employer</td>
<td>MTS_UC_ Third Party Employer.doc</td>
<td>Alfresco System -&gt; MTS -&gt; Design – Detail Design</td>
</tr>
</tbody>
</table>
5.11.8 MTS Use Case: Third Party Financial Institution

**Summary**
This component operates with the Third Party Search module to maintain information related to third party financial institutions.

**Approach**
MTS users can activate the Third Party Financial Institution pop-up from the third party search module. The popup allow user to update information on third party financial institutions.

The Third Party Financial Institution module reads and/or updates data from MTS tables as depicted in Section 3, Database of the Use Case Document.

**Conclusion:**
The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
</table>
5.11.9 MTS Use Case: Third Party Genetic Testing Lab

Summary
This component operates with the Third Party Search module to maintain information related to third party genetic testing labs.

Approach
MTS users can activate the Third Party Genetic Testing Lab popup from the third party search module. The popup allow user to update information on third party genetic testing labs.

The Third Party Genetic Testing Lab module reads and/or updates data from MTS tables as depicted in Section 3, Database of the Use Case Document.

Conclusion:
The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
</table>
5.11.10 MTS Use Case: Third Party Health Care Provider

Summary
This component operates with The Third Party Search module to maintain information related to third party health care providers.

Approach
MTS users can activate the Third Party Health Care Provider pop-up from the Third Party Search module. The popup allow user to update information on third party health care providers.

The Third Party Health Care Provider module reads and/or updates data from MTS tables as depicted in Section 3, Database of the Use Case Document.

Conclusion:
The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case: Third Party Health Care Provider</td>
<td>MTS_UC_ Third Party Health Care Provider.doc</td>
<td>Alfresco System -&gt; MTS -&gt; Design – Detail Design</td>
</tr>
</tbody>
</table>
5.11.11 MTS Use Case: Third Party Health Insurance Provider

**Summary**

This component operates with the Third Party Search module to maintain information related to third party insurance providers.

**Approach**

MTS users can activate the Third Party Health Insurance Provider popup from the Third Party Search module. The popup allow user to update information on third party health insurance providers.

The Third Party Health Insurance Provider module reads and/or updates data from MTS tables as depicted in Section 3, Database of the Use Case Document.

**Conclusion:**

The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case: Third Party Health Insurance Provider</td>
<td>MTS_UC_ Third Party Health Insurance Provider.doc</td>
<td>Alfresco System -&gt; MTS -&gt; Design – Detail Design</td>
</tr>
</tbody>
</table>

5.11.12 MTS Use Case: Third Party Law Enforcement

**Summary**

This component operates with the Third Party Search module to maintain information related to third party law enforcement.

**Approach**

MTS users can activate the Third Party Law Enforcement popup from the Third Party Search module. The popup allow user to update information on third party law enforcement.

The Third Party Law Enforcement module reads and/or updates data from MTS tables as depicted in Section 3, Database of the Use Case Document.

**Conclusion:**

The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case: Third Party Law Enforcement</td>
<td>MTS_UC_ Third Party Law Enforcement.doc</td>
<td>Alfresco System -&gt; MTS -&gt; Design – Detail Design</td>
</tr>
</tbody>
</table>

Version 1.12 06/13/2014 191
5.11.13  MTS Use Case: Third Party Probation Parole Officer

Summary
This component operates with the Third Party Search module to maintain information related to third party probation parole officers.

Approach
MTS users can activate the Third Party Probation Parole Officer popup from the Third Party Search module. The popup allow user to update information on third party probation parole officers.

The Third Party Probation Parole Officer module reads and/or updates data from MTS tables as depicted in Section 3, Database of the Use Case Document.

Conclusion:
The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case: Third Party Probation Parole Officer</td>
<td>MTS_UC_ Third Party __ Probation Parole Officer.doc</td>
<td>Alfresco System -&gt; MTS -&gt; Design – Detail Design</td>
</tr>
</tbody>
</table>
5.11.14 MTS Use Case: Third Party Registered Agent

Summary
This component operates with the Third Party Search module to maintain information related to third party registered agents.

Approach
MTS users can activate the Third Party Registered Agent popup from the Third Party Search module. The popup allows users to update information on third party registered agents.

The Third Party Registered Agent module reads and/or updates data from MTS tables as depicted in Section 3, Database of the Use Case Document.

Conclusion:
The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case: Third Party Registered Agent</td>
<td>MTS_UC_ Third Party Registered Agent.doc</td>
<td>Alfresco System -&gt; MTS -&gt; Design – Detail Design</td>
</tr>
</tbody>
</table>

5.11.15 MTS Use Case: Third Party School

Summary
This component operates with the Third Party Search module to maintain information related to third party schools.

Approach
MTS users can activate the Third Party School popup from the third party search module. The popup allows users to update information on third party schools.

The Third Party School module reads and/or updates data from MTS tables as depicted in Section 3, Database of the Use Case Document.

Conclusion:
The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case: Third Party School</td>
<td>MTS_UC_ Third Party School.doc</td>
<td>Alfresco System -&gt; MTS -&gt; Design – Detail Design</td>
</tr>
</tbody>
</table>
5.11.16 MTS Use Case: Third Party TCSE Office

**Summary**
This component operates with the Third Party Search module to maintain information related to third party TCSE offices.

**Approach**
MTS users can activate the Third Party TCSE Office popup from the third party search module. The popup allow user to update information on third party TCSE offices.

The Third Party TCSE Office module reads and/or updates data from MTS tables as depicted in Section 3, Database of the Use Case Document.

**Conclusion:**
The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case: Third Party</td>
<td>MTS_UC_ Third Party Tribal CSE Office</td>
<td>Alfresco System -&gt; MTS -&gt; Design – Detail Design</td>
</tr>
<tr>
<td>Tribal CSE Office</td>
<td>CSE Office.doc</td>
<td></td>
</tr>
</tbody>
</table>
6  Data Design

This section describes the design components of the MTS Data Model. It is the basis for the
detailed design and coding of the data definition language and the data manipulation language to
be written for the supporting database management system (DBMS). The data dictionary and
logical/ physical database design reflected in this section are system generated and reflect the
data model as of the documents generation date. The database design described below reflects
the database requirements specified in the Software Requirements Specification (SRS).

6.1  Database Standards

The database standards document provides a base set of rules that can be used as a foundation to
create the physical data model. All standards defined in the documentation will be applied to the
MTS MySQL database to during design and development.

The document is referenced in the table below.

<table>
<thead>
<tr>
<th>Track</th>
<th>Document Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database: Database Standardization</td>
<td>MTS_Database_Standards.doc</td>
<td>Alfresco System -&gt; MTS -&gt; Database</td>
</tr>
</tbody>
</table>

6.2  Database Design Based on Business Functions

See the MTS Data Model

6.3  Database Table Layouts and Relationships

See the MTS Data Model
7 System Performance

This section highlights the system performance specifications for the MTS, as defined by Section 3 of the General System Design document. It presents three architectures and the general specifications for system performance. The first architecture is intended to support a single office program with a thousand cases or less. The second architecture is intended to support a single or multi-office program with up to 20,000 cases. The third architecture is intended to support multiple programs using multiple instances of the MTS software and multiple instances of the same database on shared hardware. This architecture is expected to support programs totaling up to 20,000 cases.

MTS system performance will be determined by the limitations of the hardware. Hardware limitation will be alleviated by developing a system that takes advantage of load balancing technologies to create a highly scalable system. Scalability can be achieved through the use of: load balance-capable operating systems (OS), clustered web servers, clustered database servers, and clustered application servers. Load balancing will enable the system to use multiple servers and pool the resources together. This allows hardware to be added as needed to improve the overall performance of the system.

System availability will be improved through clustering and load balancing technologies. Clustering will prevent hardware failures from affecting system performance and uptime. A failed server will be automatically removed from the cluster and processing will take place on the remaining servers in the cluster.

Section 3.2, System Architecture outlines the system solution architecture using the following three (3) cluster models:

- Vertical Cluster (single server).
- Vertical Cluster (multiple server).
- Horizontal Cluster.

A Vertical Cluster has multiple application server instances hosted on the same physical machine (node).

A Horizontal cluster has multiple physical machines (nodes).

The System Architecture document Section 9, Size and Performance, describes how standard MTS installation will cluster physical machines to improve system performance (minimize downtime, improve response time, and maximize scalability).

The Application Cache Use Case document describes how the JBossCache framework will be implemented as a load balancer to maximize response time, among other cache capabilities.

7.1 Operating System Clustering

The underlying OS for the system will be capable of clustering or load balancing to ensure high availability and optimum system performance.
7.2 Application Server Clustering
The application server will allow multiple instantiations to be created. This will enable the system to share the application processing across multiple hardware resources increasing system performance and availability.

7.3 Database Server Clustering
The database server will provide clustering across multiple servers. Database clustering allows databases installed across multiple databases servers to appear as a single instance. Unavailable servers will not affect the clusters ability to function.
# Appendices

## Appendix A – Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS</td>
<td>Operating System</td>
</tr>
<tr>
<td>BOM</td>
<td>Bill of Materials</td>
</tr>
<tr>
<td>VO</td>
<td>Value Object</td>
</tr>
<tr>
<td>SRS</td>
<td>Software Requirements Specification</td>
</tr>
<tr>
<td>GSD</td>
<td>General System Design</td>
</tr>
<tr>
<td>J2EE</td>
<td>Java 2 Enterprise Edition</td>
</tr>
<tr>
<td>UAT</td>
<td>User Acceptance Test</td>
</tr>
<tr>
<td>UI</td>
<td>User Interface</td>
</tr>
<tr>
<td>RIA</td>
<td>Rich Internet Application</td>
</tr>
<tr>
<td>MVC</td>
<td>Model View Controller</td>
</tr>
<tr>
<td>MXML</td>
<td>Multimedia eXtensible Markup Language</td>
</tr>
<tr>
<td>CSS</td>
<td>Cascading Style Sheets</td>
</tr>
</tbody>
</table>
Appendix B – References


MTS Data Dictionary and Data Model
## Appendix C – IT Bill of Materials (BOM)

### Infrastructure Products

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Architecture Component</th>
<th>Manufacture/Vendor</th>
<th>Product Name</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Application Server software</td>
<td>JBoss</td>
<td>JBoss</td>
<td>4.05</td>
</tr>
<tr>
<td>1</td>
<td>Application Server/HTTP Server Hardware</td>
<td>TBD</td>
<td>TBD</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>HTTP Server Software</td>
<td>Apache</td>
<td>Apache</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Database Software</td>
<td>MySQL</td>
<td>MySQL</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>Database Server Hardware (MySQL)</td>
<td>TBD</td>
<td>TBD</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Version Control</td>
<td>SubVersion</td>
<td>SubVersion</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Version Control Hardware</td>
<td>TBD</td>
<td>TBD</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Development Software</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Reporting Software</td>
<td>Jasper Reports</td>
<td>Jasper Reports</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Reporting Software</td>
<td>iReport</td>
<td>iReport</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Business Process Modeling</td>
<td>JBoss</td>
<td>jBPM</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Cache</td>
<td>JBoss</td>
<td>Cache</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Developer Platform</td>
<td>Eclipse</td>
<td>Eclipse WTP</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Developer Platform Extension for JBoss</td>
<td>JBoss</td>
<td>JBoss IDE</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Subclipse</td>
<td>SubVersion</td>
<td>Subclipse</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Flex2 SDK</td>
<td>Adobe</td>
<td>Flex2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>JDK 1.5</td>
<td>Sun</td>
<td>Java</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Security Software</td>
<td>ACEGI</td>
<td>ACEGI Security</td>
<td></td>
</tr>
</tbody>
</table>
### Other Services

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Architecture Component</th>
<th>Manufacture/Vendor</th>
<th>Product Name</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Global Web Hosting</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Data Center</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>WAN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>LDAP Services</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>VPN</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>