Field Force Management Integration Interface Specification Version 1.0

Committee Specification 01

05 October 2012

Specification URIs

This version:

http://docs.oasis-open.org/ffm/FFMII-SPEC/v1.0/cs01/FFMII-SPEC-v1.0-cs01.pdf (Authoritative)
http://docs.oasis-open.org/ffm/FFMII-SPEC/v1.0/cs01/FFMII-SPEC-v1.0-cs01.html
http://docs.oasis-open.org/ffm/FFMII-SPEC/v1.0/cs01/FFMII-SPEC-v1.0-cs01.doc

Previous version:

https://www.oasis-open.org/committees/download.php/46595/FFMII-Specification-v1.0-csprd02.zip

Latest version:

http://docs.oasis-open.org/ffm/FFMII-SPEC/v1.0/FFMII-SPEC-v1.0.pdf (Authoritative)
http://docs.oasis-open.org/ffm/FFMII-SPEC/v1.0/FFMII-SPEC-v1.0.html
http://docs.oasis-open.org/ffm/FFMII-SPEC/v1.0/FFMII-SPEC-v1.0.doc

Technical Committee:

OASIS Field Force Management (FFM) TC

Chair:

Thinh Nguyenphu (thinh.nguyenphu@nsn.com), Nokia Siemens Networks

Editor:

Thinh Nguyenphu (thinh.nguyenphu@nsn.com), Nokia Siemens Networks

Additional artifacts:

This prose specification is one component of a Work Product which also includes:

- WSDL files: http://docs.oasis-open.org/ffm/FFMII-SPEC/v1.0/cs01/wsdl/
- XML Schema files: http://docs.oasis-open.org/ffm/FFMII-SPEC/v1.0/cs01/wsdl/schemas/

Declared XML namespaces:

The following namespaces are declared in Section 9.1.1:

- http://docs.oasis-open.org/ffm/ns/v1.0/ws/implementation
- http://docs.oasis-open.org/ffm/ns/v1.0/ws/manager
- http://docs.oasis-open.org/ffm/ns/v1.0/common/api
- http://docs.oasis-open.org/ffm/ns/v1.0/common/model
- http://docs.oasis-open.org/ffm/ns/v1.0/system/info/api
- http://docs.oasis-open.org/ffm/ns/v1.0/system/info/model
- http://docs.oasis-open.org/ffm/ns/v1.0/system/capability/api
- http://docs.oasis-open.org/ffm/ns/v1.0/system/capability/model
- http://docs.oasis-open.org/ffm/ns/v1.0/wrm/api
- http://docs.oasis-open.org/ffm/ns/v1.0/wrm/model
- http://docs.oasis-open.org/ffm/ns/v1.0/firm/api
- http://docs.oasis-open.org/ffm/ns/v1.0/firm/model
- http://docs.oasis-open.org/ffm/ns/v1.0/rdm/api
- http://docs.oasis-open.org/ffm/ns/v1.0/rdm/model

- http://docs.oasis-open.org/ffm/ns/v1.0/rdm/model/profile

Abstract:

This document describes the Field Force Management Integration Interface (FFMII). FFMII provides a flexible interface between *Enterprise Resource Management System* (ERMS) and *Field Force Management System* (FFMS). The role of ERMS is to take a holistic view at work scheduling and resource allocation from the corporate point of view. For this purpose it needs to manage individual units of work performed without strong supervisory guidance (*Field Work*) in a way aligned with the business objectives of the company. The role of FFMS is to communicate with available set of workers (*Field Force*) and to provide individual workers (*Assignees*) performing Field Work with technical means of accessing information about and sending feedback on work assigned to them. While ERMS defines the structure, content and resource allocation of dispatched work, FFMS is responsible for communicating that information to the field and enforcing any specified constraints on user provided feedback. For facilitating structured communication between ERMS and FFMS in heterogeneous scenarios, FFMII defines flexible mechanisms that enable *Work Request* modeling (data content, workflow), exchange, work history collection, and collection of data from the field. Information carried with work requests, work request structure (work-flow, schedule) and data to be collected can all be defined dynamically 'as data'. This data driven architecture makes FFMII very flexible and adaptable to numerous industries. Additional FFMII capabilities include *Field-Initiated Requests* that facilitate structured requests and reporting information outside the usual work flow, such as reporting absence, requesting additional work, or providing a sales lead. Reference Data Management provides means to establish custom data repositories with arbitrary content. This enables e.g. input value selection, content validation, or delivery of documents to Field Force. Furthermore, flexible Integration Topologies are supported. Further flexibility is provided by scalability of FFMII itself: Basic features of FFMII are mandatory, and some features are optional. This allows both simple basic implementations and a range of more complete implementations.

Status:

This document was last revised or approved by the OASIS Field Force Management (FFM) TC on the above date. The level of approval is also listed above. Check the "Latest version" location noted above for possible later revisions of this document.

Technical Committee members should send comments on this specification to the Technical Committee's email list. Others should send comments to the Technical Committee by using the "Send A Comment" button on the Technical Committee's web page at https://www.oasis-open.org/committees/ffm/.

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the Technical Committee web page (https://www.oasis-open.org/committees/ffm/ipr.php).

Citation format:

When referencing this specification the following citation format should be used:

[FFMII-SPEC-v1.0]

Field Force Management Integration Interface Specification Version 1.0. 05 October 2012. OASIS Committee Specification 01.
http://docs.oasis-open.org/ffm/FFMII-SPEC/v1.0/cs01/FFMII-SPEC-v1.0-cs01.html.

Notices

All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the "OASIS IPR Policy"). The full Policy may be found at the OASIS website.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published, and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to OASIS, except as needed for the purpose of developing any document or deliverable produced by an OASIS Technical Committee (in which case the rules applicable to copyrights, as set forth in the OASIS IPR Policy, must be followed) or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

OASIS requests that any OASIS Party or any other party that believes it has patent claims that would necessarily be infringed by implementations of this OASIS Committee Specification or OASIS Standard, to notify OASIS TC Administrator and provide an indication of its willingness to grant patent licenses to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification.

OASIS invites any party to contact the OASIS TC Administrator if it is aware of a claim of ownership of any patent claims that would necessarily be infringed by implementations of this specification by a patent holder that is not willing to provide a license to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification. OASIS may include such claims on its website, but disclaims any obligation to do so.

OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS' procedures with respect to rights in any document or deliverable produced by an OASIS Technical Committee can be found on the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this OASIS Committee Specification or OASIS Standard, can be obtained from the OASIS TC Administrator. OASIS makes no representation that any information or list of intellectual property rights will at any time be complete, or that any claims in such list are, in fact, Essential Claims.

The name "OASIS" is a trademark of OASIS, the owner and developer of this specification, and should be used only to refer to the organization and its official outputs. OASIS welcomes reference to, and implementation and use of, specifications, while reserving the right to enforce its marks against misleading uses. Please see http://www.oasis-open.org/policies-guidelines/trademark for above guidance.

Table of Contents

1 Document Scope

This document describes the *Field Force Management Integration Interface* (FFMII) comprising of the following topics:

- Foundation: Context definitions, Domain Model, Architecture, Core Data Model.
- Identity and Capabilities Discovery
- Principal Functional Areas: Work Request Management, Reference Data Management, Change Notification Management, and Field-Initiated Request Management.

Figure 1 outlines the structure of the FFMII specification:

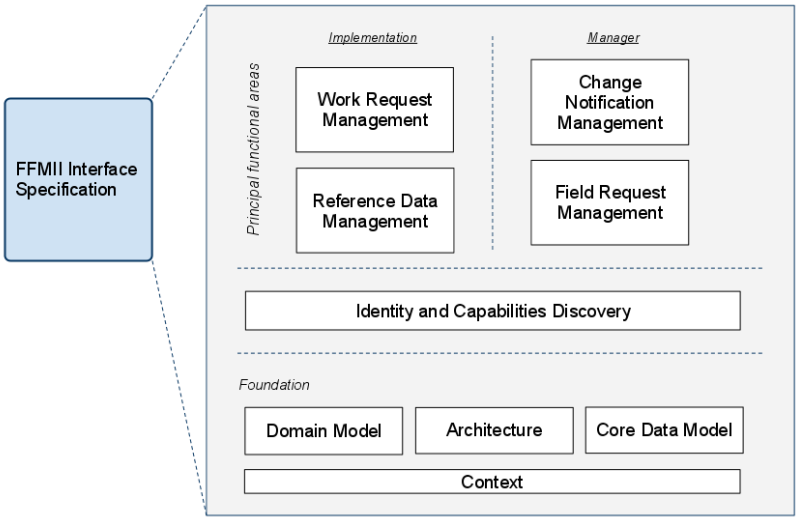


Figure 1: FFMII Interface Specification Overview

Field Force Management Integration Interface Requirements are specified in [FFMII-REQ]

2 References

2.1 Normative References

[FFMII-WSDL]	Field Force Management Integration Interface Specification WSDL. Location provided in “Additional artifacts” section in header material of this document.
[KML]	Open GeoSpatial Consortium OGC 07-147r2, "OGC KML", http://www.opengeospatial.org/standards/kml, version 2.2.0, April 2008
[ISO-639]	ISO 639-1:2002, Part 1: Alpha-2 Codes, "Codes for the representation of names of languages", http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=22109
[ISO-3166]	ISO 3166-1:2006, Alpha-2 Country Codes, http://www.iso.org/iso/country_codes.htm
[ISO/IEC 8859-1]	ISO/IEC 8859-1:1998, Information technology — 8-bit single-byte coded graphic character sets — Part 1: Latin alphabet No. 1 http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=28245
[RFC2046]	Freed, N. and Borenstein, N. Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types, http://tools.ietf.org/html/rfc2046**,** IETF RFC 2046, November 1996
[RFC2119]	S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*, http://www.ietf.org/rfc/rfc2119.txt, IETF RFC 2119, March 1997.
[RFC2616]	IETF RFC 2616 “Hypertext Transfer Protocol – HTTP/1.1”, R. Fielding, June 1999, URL: http://www.ietf.org/rfc/rfc2616.txt
[RFC2818]	IETF RFC 2818 “HTTP Over TLS”, Rescorla, E., May 2000, URL: http://www.ietf.org/rfc/rfc2818.txt
[Schema2]	P. V. Biron et al. XML Schema Part 2: Datatypes. World Wide Web Consortium Recommendation, May 2001. See http://www.w3.org/TR/xmlschema-2/
[SOAP]	SOAP Version 1.2 Part 1: Messaging Framework (Second Edition), Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, Henrik Frystyk Nielsen, Anish Karmarkar, Yves Lafon, Editors. World Wide Web Consortium, 27 April 2007. This version is http://www.w3.org/TR/2007/REC-soap12-part1-20070427. The latest version is available at http://www.w3.org/TR/soap12-part1/.
[UML]	*OMG Unified Modeling Language Specification,* Version 1.5. March 2003. http://www.omg.org/spec/UML/1.5/PDF/. Non-normative note: Version 1 (e.g. latest version 1.5) is sufficient for FFMII. http://www.uml.org/ provides access to the various versions of UML, including the newest one.
[WSS]	*Web Services Security: SOAP Message Security Version 1.1.1*. 18 May 2012. OASIS Standard. http://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-os-SOAPMessageSecurity.pdf.
[WSS-UTP]	*Web Services Security Username Token Profile Version 1.1.1*. 18 May 2012. OASIS Standard. http://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-os-UsernameTokenProfile.pdf.

2.2 Non-Normative References

[FFMII-REQ] *Field Force Management Integration Interface Requirements Version 1.0*. 05 October 2012. OASIS Committee Note 01. http://docs.oasis-open.org/ffm/FFMII-REQ/v1.0/cn01/FFMII-REQ-v1.0-cn01.html**.**

[UML-informal] *Allen Holub's UML Quick Reference,* provides an easy to access summary to UML notation. http://www.holub.com/goodies/uml/

3 Definitions and Conventions

3.1 Conventions

3.1.1 Usage of Requirement Level Keywords

This specification uses normative text. The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this specification are to be interpreted as described in [RFC2119]:

> …they MUST only be used where it is actually required for interoperation or to limit behavior which has potential for causing harm (e.g., limiting retransmissions)…

These keywords are thus capitalized when used to unambiguously specify requirements over protocol and application features and behavior that affect the interoperability and security of implementations. When these words are not capitalized, they are meant in their natural-language sense.

3.1.2 Notation for data structure declaration

Throughout this document, structured data types of fixed content (classes) are declared using the following convention:

< Class Name >			
Property	**Type**	**M/O**	**Description**
<property –name-1>	<property-data-type>	<optionality indicator>	< notes and descriptions >
<property –name-2>	<property-data-type>	<optionality indicator>	< notes and descriptions >

The <Class Name > label represents the name of the class being introduced. The column “Property” introduces names of individual constituents of the class in question, and column “Type” the type of each property. Type can be reference to any data type specified as part of the Core Data Model (6.3), or any other class introduced in this specification.

Property is regarded mandatory if its optionality indicator is set to “**M**”, or optional if set to “**O**”.

3.1.3 Notation for diagrams

Throughout this document, diagrams illustrate and specify concepts of the FFMII interface, interaction patterns etc. UML notation is applied [UML]. Non-normative note: There are many guides and summaries of UML notation that may be quicker to access than the UML standard. For example, see [UML-informal].

3.1.4 Normative and Non-Normative parts of the document

All sections and appendixes, except “Document Scope” are normative, unless they are explicitly indicated to be informative.

3.2 Definitions

Activity	Activity represents a distinct part of the Task the Assignee is requested to accomplish. An Assignee can carry at most single Activity at any given time while being allowed to switch between Activities under certain conditions. Progress and the Activity completion status form the basis for reporting to the upper levels of enterprise resources management. An Activity consists of one or more Steps, possibly forming a controlled micro-flow within the Activity itself.
Activity State Model	An Activity State Model is a combination of States, Steps and Actions, defined for each Activity. The Work flow of a Task is specified through Activity State Models of each included Activity. Activity State Models are defined as data to remain neutral with respect to types of Task a Work Request that can be represented.
Assignee	Assignee is a human resource performing field operations
Data Form	Data Forms are used to model dynamically specified structured information for the purpose of conveying e.g. Work Request header, overview and instructions, user input and Field Initiated Request contents They consist of Data Form Elements such as individual Data Fields, Data Attachments and Data Matrixes, with means to group the Data Elements.
ERMS	Enterprise Resource Management System (ERMS) refers to one or more software components collectively responsible for assignment of resources into company business operations, including work planning, execution and exception handling
Field Force	Field Force refers to a group of Assignees to whom Work Requests are delivered using FFMII interface
FFMII	The Field Force Management Integration Interface (FFMII) provides a flexible interface between ERMS and FFMS for the purpose of Work Request modeling, exchange, and collection of data from the field
FFMS	Field Force Management System (FFMS) refers to one or more software components collectively responsible for efficient communication with the Field Force
Field Work	Field Work refers to work that is expected to be conducted by an individual (or a group of closely co-operating individuals) without need for strong supervisory guidance. Units of Field Work are informally referred to as Tasks, and a unit of Field Work is modeled as a Work Request associated with a Work Type Specification, see [FFMII-SPEC] Section 5.1.1 and Task below.
Field-Initiated Request	Interaction pattern used for relying requests initiated by an Assignee in the field and targeting specific functionality inside of Manager. Field-Initiated Interactions are either Topical Notifications or Topical Inquiries, depending on whether any response is expected for the request in question or not.
Implementation	Implementation is the role of a software system communicating with ERMS through the FFMII interface. Implementation refers to parts of FFMS implementing behavioral patterns specified by this document and exposed through the Interface. For example, an Implementation submits Status Change Notifications and creates Field-Initiated Requests though the FFMII interface.
Interface	Interface refers to the Field-Force Management Integration Interface (FFMII) specified throughout this document, unless stated otherwise
Manager	Manager is the role of a software system communicating with FFMS through the FFMII interface. Manager refers to parts of ERMS implementing behavioral

	patterns specified by this document and exposed through the Interface. For example, a Manager creates, updates, and queries units of Field Work though the FFMII interface.
Peer	Peer is an Assignee cooperating in delivery of particular Field Work with one or more other Assignees
Reference Data	Implementations MAY offer system and/or custom repositories that store Reference Data. System repositories provide access to selected data on Implementation side, such as Assignee identities (Users), and reusable types of Work Requests and Field-Initiated Requests. Custom repositories can store arbitrary content. Content of such repositories is commonly denoted as "Reference Data", Custom Reference Data MAY be used for input value selection, lookup of display values or content validation in Work Requests.
State	An Activity declares the set of available States of that Activity. Each Step is associated with a specific State. Each State belongs to exactly one Status category such as 'Active' or 'Closed' (see Section 5.1.2.4), and MAY be associated with one of the predefined (e.g. 'OnSite') or custom Status Indicators (see Section 5.1.2.4).
Step	Step describes a single recorded Action to be taken by the Assignee while performing a certain Activity. A Step may carry additional detailed instructions and it may require user input. In its simplest form, a Step can be merely an acknowledgement of certain part of the Activity being completed.
Task	Task refers to a well-specified group of Activities performed by one or more Assignee(s). Task is not a formal concept in FFMII. FFMII represents Tasks as Work Requests associated with a Work Type Specification. Such a Task may be completely independent of other Tasks, or be part of a larger project.
Topical Inquiry	Type of Field-Initiated Interaction, for which data is expected as an asynchronous response
Topical Notification	Type of Field-Initiated Request, for which no data is expected as a response
Work Request	Work Request is a structured representation of a Task, including basic description of the Task, specification of all included Activities, details of those Activities, Task and Activity completion status data, and all information necessary for status reconciliation in ERMS. Specification of a Work Request MUST contain sufficient amount of information necessary for accomplishing specified work in self-guided way, within expected time window and resulting into desired outcome.
Work Request Status Change Notification	Asynchronous message sent by Implementation to Manager whenever State of Work Request changes due to Assignee actions
Work Request Status Record	Work Request Status Record stores State changes of a Work Request after it has been received by the Implementation. It contains exactly one Tasks Status Record that gives an overview of the status of the Task, and zero or more Change History Entries that record Activity State changes, and Data Change History Entries that reflect changes in updateable Data Elements.
Work Type Specification	A Work Type Specification (WTS) specifies content and structure of a Work Request. These include Activities, their work flow, and the structure of associated data content. Activities can be performed in sequence or parallel, and they may have dependencies on each other. A Work Request provides Task instance specific data that fills in the structural definition, such as values

	for data elements.

3.3 Abbreviations

ERMS	See Section 3.2 Definitions
FFMII	See Section 3.2 Definitions
FFMS	See Section 3.2 Definitions
FIR	Field-Initiated Request
UTC	Coordinated Universal Time
WR	Work Request
WTS	Work Type Specification

4 FFMII Interface Context

4.1 System Context

Field Force Management Integration Interface (FFMII) establishes a logical link between two types of functional components:

- *Enterprise Resource Management System* (ERMS) represents a stereotype of a functional component that takes a holistic view at work scheduling and resource allocation from the corporate point of view. The ultimate purpose of an ERMS is to manage Field Work in a way aligned with the business objectives of the company. In doing so, Manager component of an ERMS uses services of FFMS to communicate with the Field Force.

- *Field Force Management System* (FFMS) communicates with Field Force, providing Assignees with technical means of accessing information about and sending feedback on work assigned to them. While ERMS defines the structure, content and resource allocation of dispatched work, FFMS is responsible for communicating that information to the field and enforcing any specified constraints on user provided feedback. The Implementation component realizes the FFMII interface for the services provided by FFMS.

Figure 2 outlines the relative positioning of ERMS, FFMS and FFMII. ERMS creates, generates updates, and queries units of Field Work using services provided by FFMS through the FFMII interface. Throughout the rest of this document, ERMS is therefore referred to as “Manager”, and FFMS as “Implementation”. However, from system integration point of view, the FFMII interface also includes services that ERMS MAY implement to receive notifications and Field-Initiated requests from FFMS.

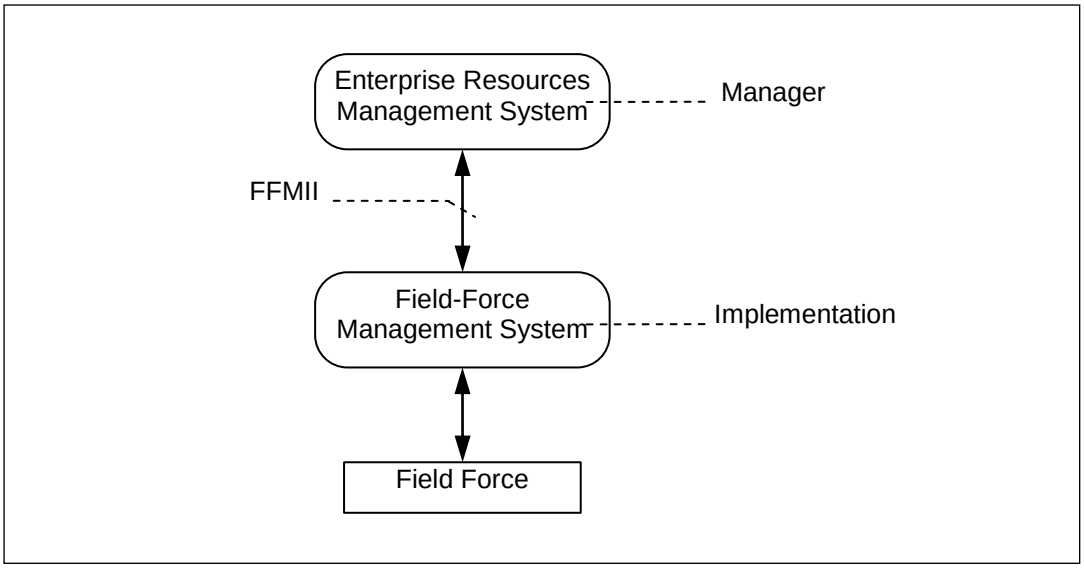


Figure 2: Field Force Management Integration Interface Overview

4.2 Interaction Styles

4.2.1 Manager-Initiated Interactions

FFMII employs *Manager-Initiated Interactions* concept for exchanging Work Request and status updates. Using this interaction pattern, a Manager submits Work Requests to an Implementation, and MAY update or cancel those as needed. A Manager MAY also poll for status changes of submitted Work Requests as necessary. An Implementation MAY, if adequately equipped, actively send Work Request Status Change Notifications towards the Manager whenever State of Work Request changes due to Assignee actions.

Figure 3 provides an overview of Manager Initiated Interaction pattern:

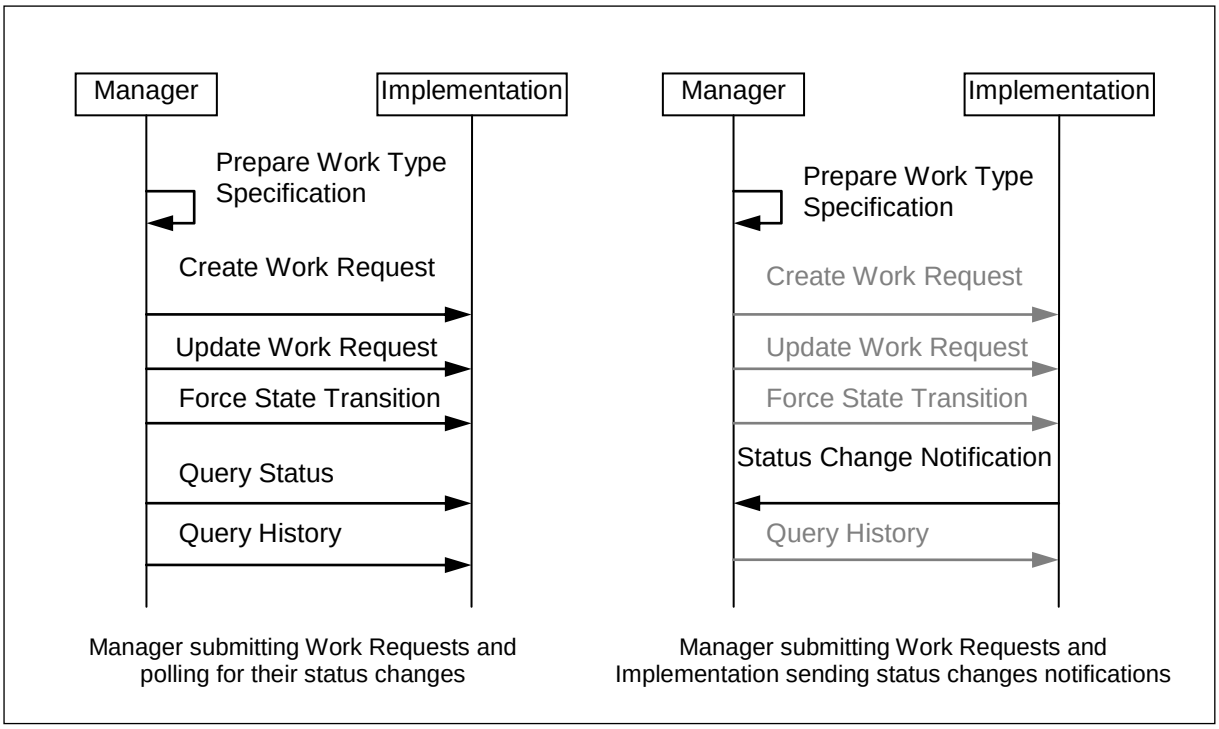


Figure 3: Manager-Initiated Work Requests

An Implementation MUST support Work Request status polling and it SHOULD support sending of status change notifications.

4.2.2 Field-Initiated Interactions

FFMII employs *Field-Initiated Interactions* concept for relying requests initiated by an Assignee in the field and targeting specific functionality inside of ERMS. Such interactions can be used for triggering ERMS actions in the context of specific Work Request, or as request not related to any Work Request in particular.

Field-Initiated Requests take form of either Topical Notifications or Topical Inquiries, depending on whether response data is expected for the request in question. Responses to Topical Inquiries are asynchronous, whereby Implementation polls for responses related to the requests it has submitted. Implementation MAY also poll for responses to several Field-Initiated Requests at once.

Figure 4 describes Topical Notifications and Topical Inquiries from the data type interaction point of view:

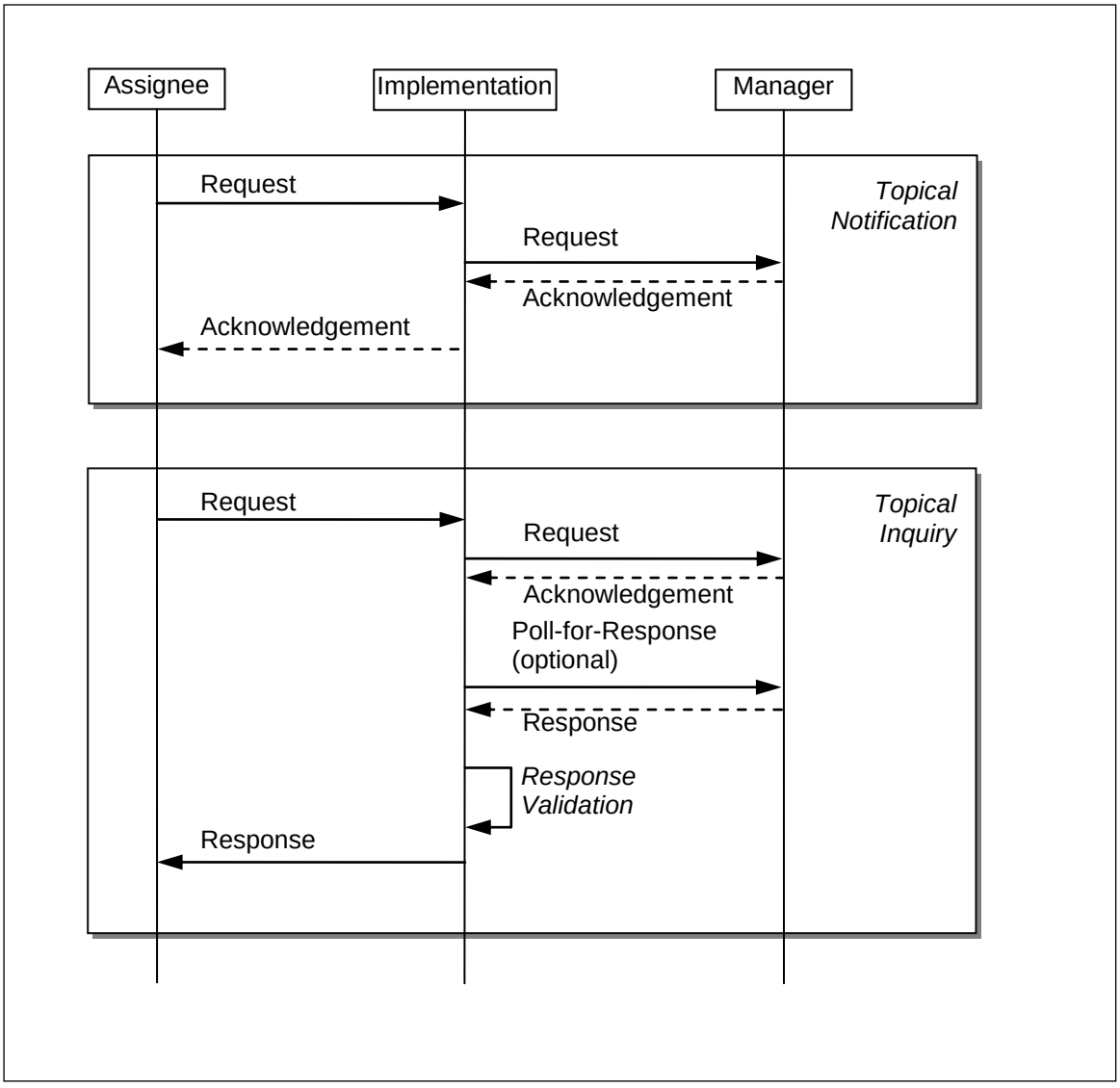


Figure 4: Field-Initiated Interactions

Note: In the FFMII context, the ultimate receiver of Field-Initiated Requests is the Manager. However, the Manager may also act as a relay agent forwarding requests (in their direct or modified form) to another integrated system, and relaying responses back to Implementation.

4.3 Integration Topologies

FFMII interface offers several ways to integrate Managers and Implementations with each other:

- **Simple topology:** a single Manager and a single Implementation interacting
- **Distributed work realization:** A single Manager interacting with several Implementations for communicating with distinct groups of field personnel
- **Shared Field Force:** multiple Managers interacting with a single Implementation

Figure 5 outlines principal differences between each type of integration scenario:

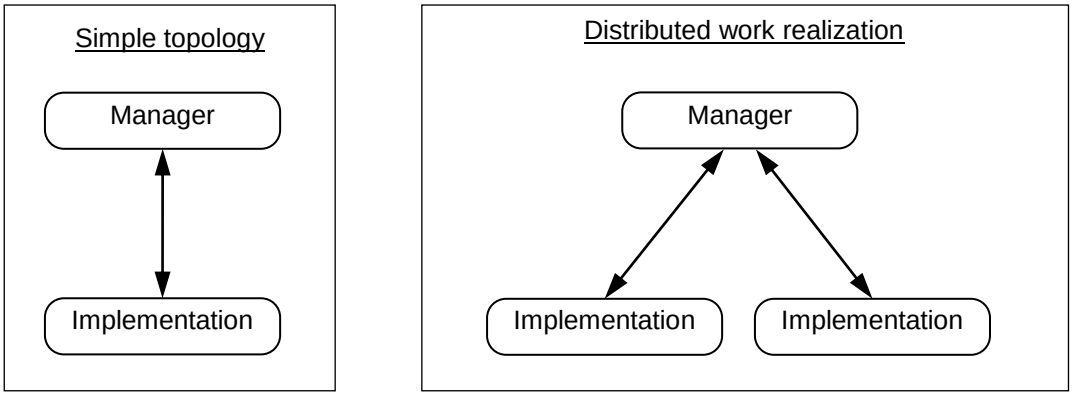


Figure 5: Simple and Advanced Integration Topologies

In Figure 5 the arrows between Managers and Implementations represent logical links between computing systems of specific roles and different *system identity*. On the network level, Managers and Implementations MAY access their counter-parts through several networking end-points if needed.

Individual integration topologies MUST not exclude each other. For example, a single Manager might be sharing a specific Implementation with another Manager for some Work Requests ("shared Field Force" scenario), while at the same time deploying other Work Requests to another Implementation ("distributed work realization"). FFMII interface also does not impose any restrictions on the number of Implementations a Manager can be integrated with, and vice-versa.

Figure 6 shows an example of a multi-paradigm integration topology featuring several Managers and several Implementations simultaneously:

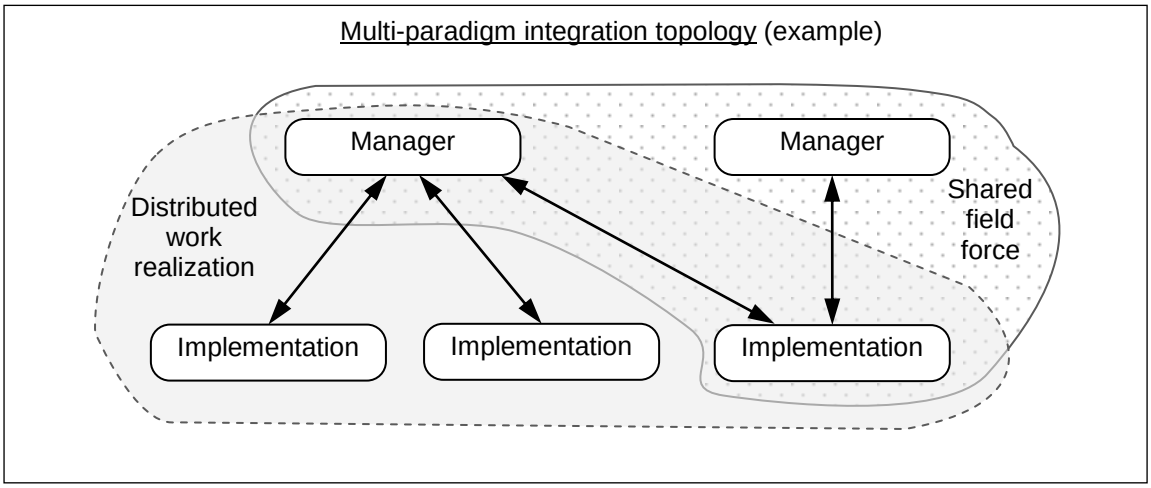


Figure 6: Multi-paradigm Integration Topology

4.3.1 Multi-tenancy support

Managers are identified through credentials included with every request they issue. When multiple Managers access the same instance of Implementation, the instance offers a private data view to each Manager, except for:

- data explicitly exposed for sharing by one of the Managers,
- data exposed for sharing by local configuration of the Implementation, and
- data defined as shared by this Interface specification

Whenever multiple Managers communicate with the same Implementation using the same credentials, they are regarded as a single Manager (logical data type) accessing the Implementation from several physical access points.

5 FFMII Interface Domain Model

5.1 Field Work

5.1.1 Overview

Manager produces series of self-contained Work Requests representing Tasks related to Field Works. Each Work Request is to be performed by one or more Assignees belonging to the addressable Field Force. A Manager communicates with one or more Implementations over the FFMII interface to make the Work Requests accessible to corresponding Assignees.

Figure 7 below outlines relations between Manager, Implementation, Field Force, Assignees and high levels structure of Work Requests representing the Tasks.

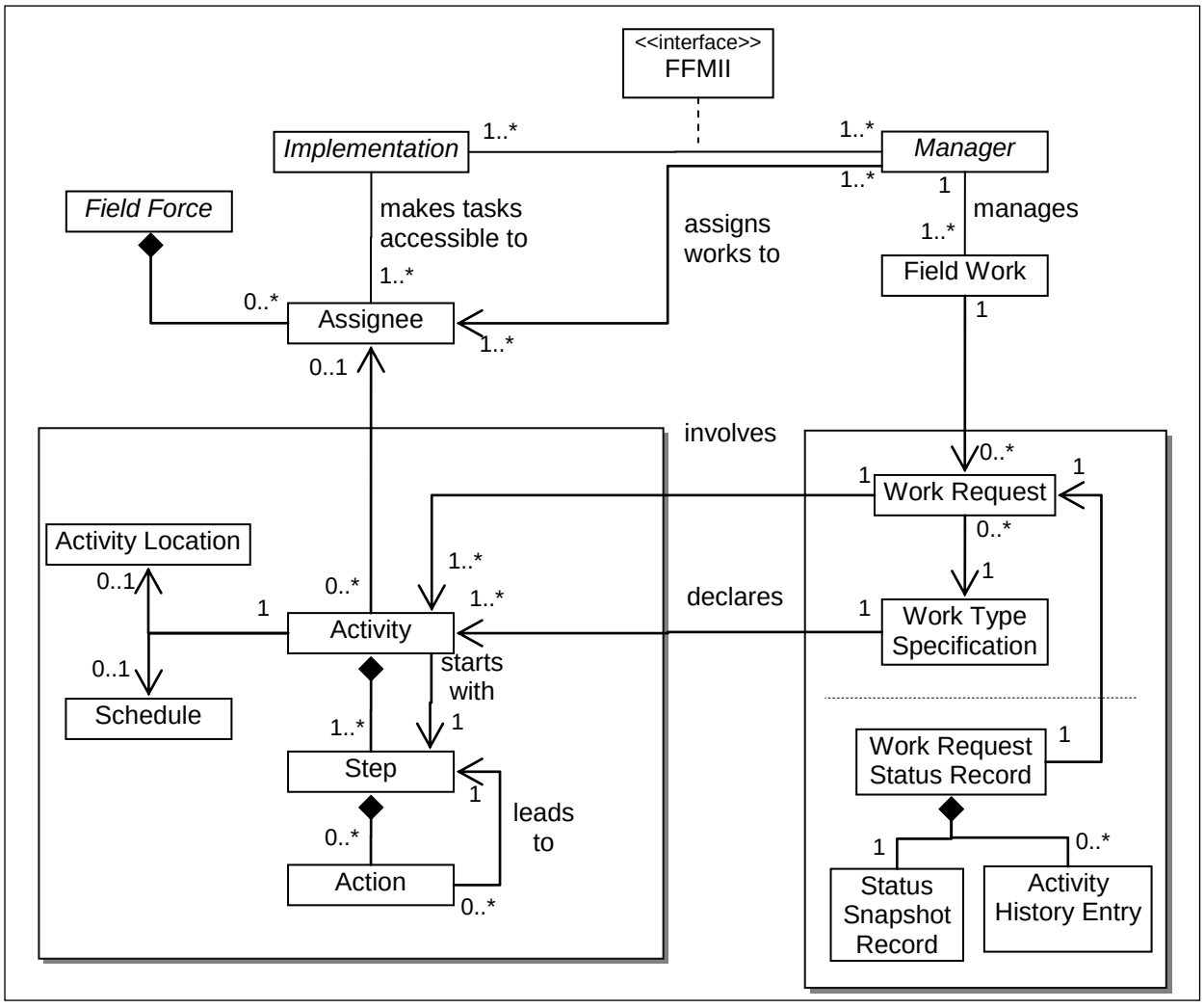


Figure 7: FFMII Interface Domain Model

A Task is described by a Work Request associated with a Work Type Specification. The Work Type Specification provides the structural definition of the Task. The Work Type Specification specifies the

associated Activities, their work flow, and the structure of associated data content. A Work Request provides Task instance specific data that fills in the structural definition, such as values for Data Elements. Several Work Requests may share the same Work Type Specification if the related Tasks have the same structure

A Work Type Specification MUST specify one or more Activities involved with the particular type of work. The Work Request provides Task specific data for the Activities. Each Activity MAY be associated with a specific Location and MAY be constrained by a Schedule. The Activity MUST be further divided into one or more Steps describing the flow of work. The Steps do not have to be sequential. One Step MUST be designated as the initial Step. Initial Step of each Activity is entered into as soon as the Activities are instantiated in the Implementation.

Each Activity MAY be associated with one Assignee who is responsible for performing it. Activities without an Assignee make it possible to create Work Requests before knowing who will perform the corresponding Activities. An Activity MUST be assigned to a specific Assignee before the Assignee can perform Actions in context of the Activity. The Assignee invokes Actions to report work progress. Each Step MAY have any number of Actions that define the possible transitions from one Step to another.

An Implementation MUST maintain a Work Request Status Record for each Work Request. A Work Request Status Record consists of a Status Snapshot Record collecting current states of Work Request and Activity and collection of Activity History Entries recording executed Actions and data supplied by the Assignee. See Section 5.1.6 for a detailed specification of Work Request Status Record.

5.1.2 Work Type Specification

5.1.2.1 External and In-lined Work Type Specifications

A *Work Type Specification* (WTS) describes content and structure of a Work Request. A Work Type Specification MAY be included as part of the Work Request itself (*in-lined WTS*), or declared though a reference to an entry in Work Type Repository managed by Implementation (*external WTS)*.

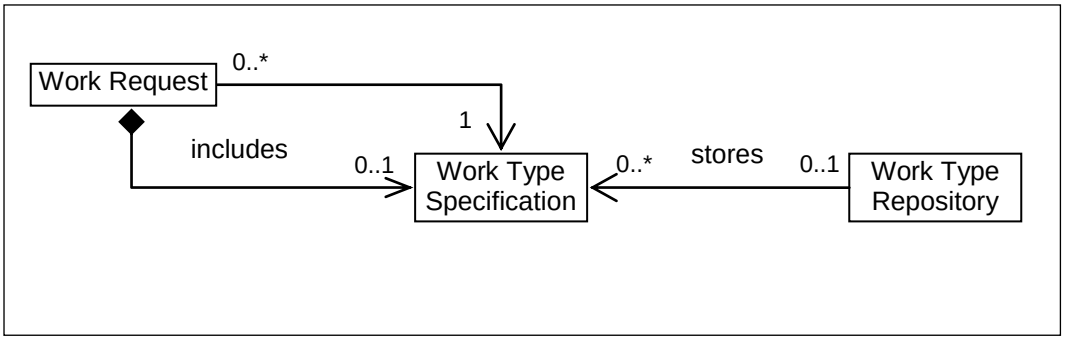


Figure 8: Work Type Specification

An external Work Type Specification MAY be shared among any number of Work Requests, while an in-lined Work Type Specification is not visible outside of the enclosing Work Request.

5.1.2.2 Work Type Specification structure

Work Type Specification divides into several parts as per the Figure 9:

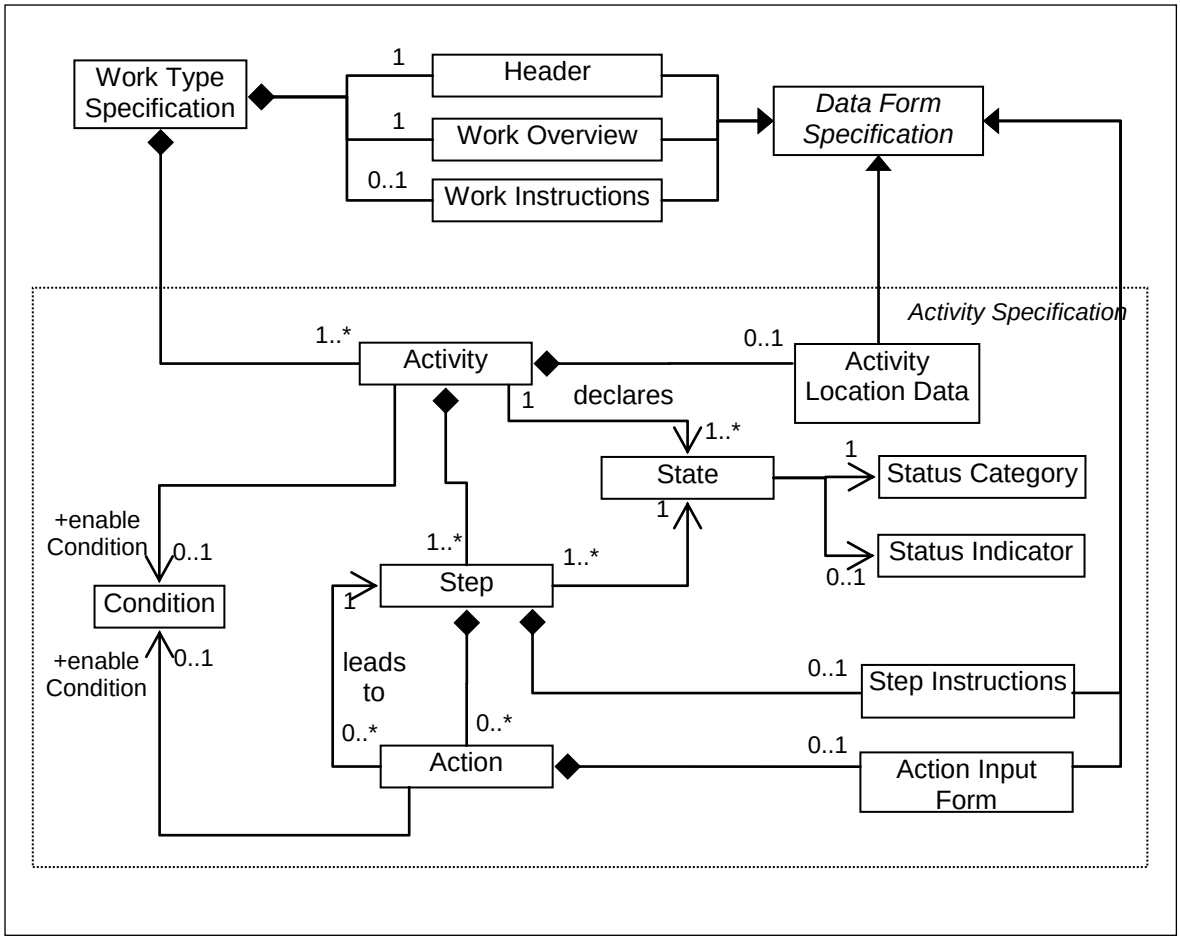


Figure 9: Work Type Specification Structure

A Work Type Specification MUST include a number of mandatory, and MAY include several optional information elements modeled as *Data Forms*. Work Request information elements are further discussed in Section 5.1.2.3, while Data Dorms are discussed in Section 5.1.4.

A Work Type Specification MUST prescribes one or more Activities with optional Data Form for Activity Location Data. Activities MAY be indicated as so-called Supplementary Activity (see later part of this section for details). A Work Type Specification MUST includes at least one Activity that is not indicated as Supplementary.

An Activity MUST consist of one or more Steps. Each Step defines zero or more Actions. An Action MAY require user input, and it MUST specify the target Step the Activity is transited to upon Action completion.

Each Step MUST be associated with a specific State. An Activity MUST declare one or more States. Each State MUST belong to exactly one Status category (see Section 5.1.2.4), and MAY be associated with one of the predefined Status Indicators (see Section 5.1.2.4). A single State MAY be shared among several Steps. In this way, for example, Work Requests MAY have several Steps that are collectively regarded as “Suspended” State, even if each Step might be carrying different display information and provide different exit paths (transitions to other Steps).

Steps, Actions and States form an Activity-level logical state model. Within this model, each Activity MUST have exactly one *initial Step*, and thereby also State. The final transition of the Activity State Model will be a transition to a State associated with Status Category “Closed” (see Section 5.1.2.4) or asserting Administrative Closing Status for the Work Request (see Section 5.1.5).

An Action MAY have an Enable Condition. If Enable Condition is False, then the Action is not available to the Assignee.

An Activity MAY have an Enable Condition. If Enable Condition is False, then the Activity is not available to the Assignee. Enable Conditions on Activity and Action MAY be used to enforce dependencies on specific Activity States (see Section 5.1.2.5 for more details).

Supplementary Activity

Any Activity in a Work Type Specification MAY be declared as so-called *Supplementary Activity*. A Supplementary Activity is not directly involved with performing some specific part of the associated Task but provides Actions (*Supplementary Actions*) that are related to the Task as a whole, such as logging a note, reporting change in the requested delivery time or providing Task completion report.

With regards to Work Request status and State changes the Supplementary Activity and the associated state model behaves just like any other Activity. However, the Implementation SHOULD present the Supplementary Activity and the associated Supplementary Actions as being related to the Task as a whole rather than being part of specific Activity only.

A potential use of multiple Supplementary Activities is to group related Supplementary Actions such as time-keeping Actions, recording notes, controlling equipment. This can be used, for example, for user interface purposes.

5.1.2.3 Work Request Information Elements

A Work Request contains several information elements that specify data structures used in various contexts. These information elements are modeled as Data Forms.

A Data Form MUST includes one or more Data Elements. A Data Element may be used for displaying data or requesting user input. For more details regarding Data Form specification refer to Section 5.1.4.

Purpose and intended usage of each Data Form defined in Work Type Specification is described below:

Header [Mandatory]
: Specifies terse identification information that allows an Assignee to identify a particular Work Request. Header information is intended to be used when an Assignee needs to choose between or otherwise identify several Work Requests, such as in list or schedule views. A Header typically includes a title or an identifier for the Work Request and possibly some other key information, such as an address or a customer name.

Work Overview [Mandatory]
: Specifies overview information describing the associated Task to the Assignee on a general level. Work Overview typically includes information about the kind of work involved, when and where the work needs to be performed and contact information. However, overview does not need to include all the details the Assignee may need to know when actually performing the Task.

Work Instructions [Optional]
: Specifies detailed instructions and other information the Assignee will need to complement the Work Overview for performing the associated Task. In addition to specific instructions, this form may also include other detailed information, such as related documentation, fault history of the associated component or the change history of the Work Request.

Step Instructions [Optional]
: Specifies detailed instructions and other information the Assignee will need for a particular Step of an Activity, if any, to complement Work Overview and Work Instructions. Step Instructions is always associated with a specific Step.

Activity Location Data [Optional]
: Specifies any special information the Assignee will need to reach the location associated with a particular Activity (Activity Location) in addition to overview information. Activity Location Data is specific to the Activity Location associated with a particular Activity and it typically includes information about special access restrictions or access procedures, and possibly detailed instructions for finding the Activity Location.

Action Input Form [Optional]

Specifies information the Assignee is required to provide when performing a particular Action. This form is primarily used for requesting input from the Assignee, however it MAY contain display elements as well.

5.1.2.4 Activity State Model

A combination of States, Steps and Actions form an *Activity State Model*. FFMII interface does not prescribe or imply usage of any specific Activity State Model in order to remain neutral with respect to types of Task a Work Request may represent. As an implication of this design decision, a Manager MUST specify an Activity State Model as part of the Work Request's Work Type Specification, and an Implementation MUST adhere to the specified Activity State Model.

Figure 10 is an example illustrating the relationship of Steps, Actions and States within an Activity. Each Step in the figure is related to some progress made by the Assignee or other resources and/or processes supporting the Assignee's work. Each Step is also associated with a specific State. Each Action in the figure leads from a Step to some other Step, in the same State or in some other State. Status Categories are marked in double angle brackets (example: "<<Open>>") and the Status Indicators are marked below each Status Category. Status Indicators starting with "X-" denote Implementation-specific indicators.

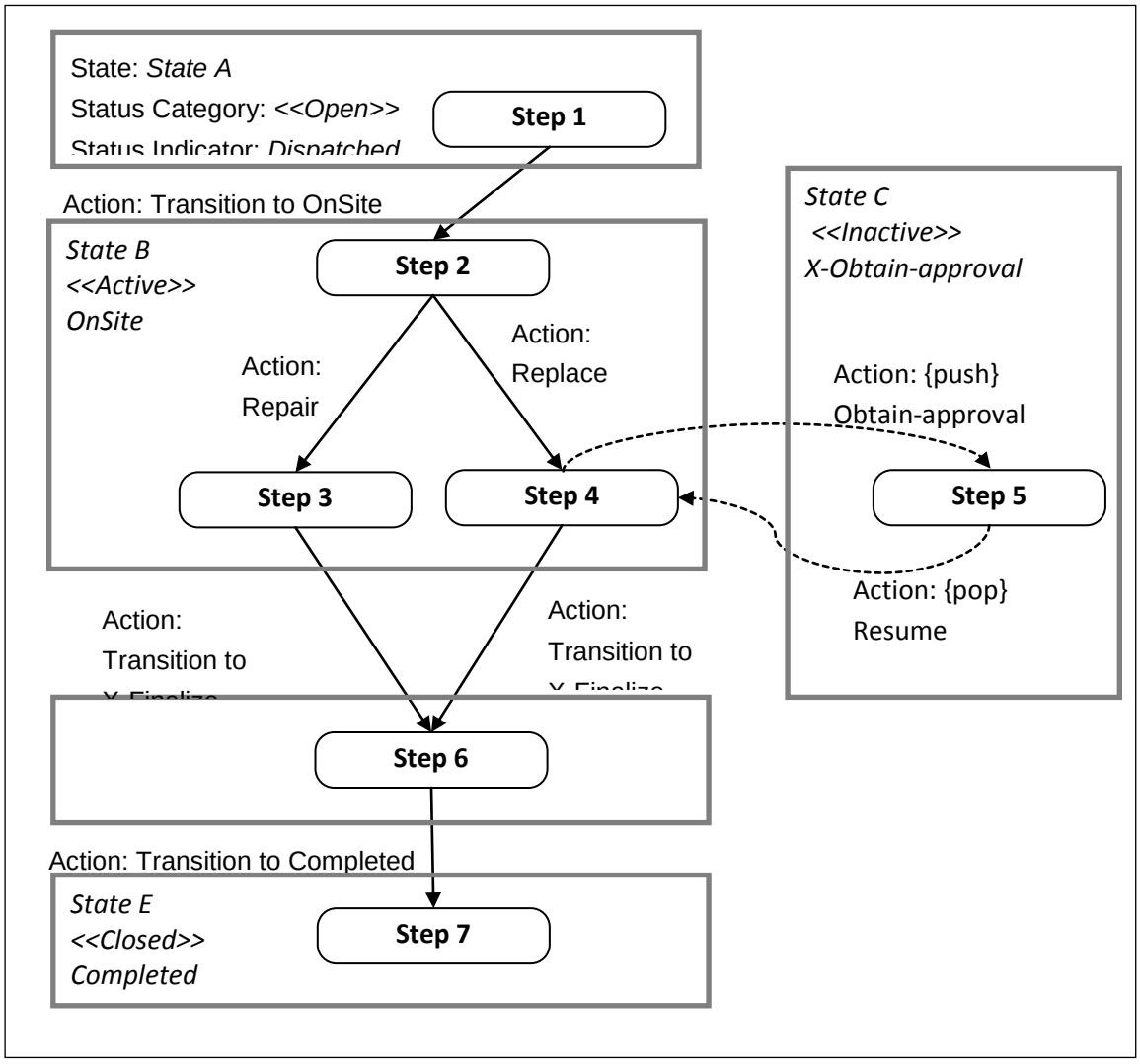


Figure 10: Relationship of Steps, Actions and States within an Activity

In this example, the OnSite State requires the Assignee to decide whether the Task may be completed by repairing the customer's equipment, or whether it is necessary to replace the equipment with a new unit. Therefore there are two possible Actions leading from Step 2, and both of them are enabled so that the Assignee may select either of them (enabling conditions aren't visualized in this diagram). If the Assignee chooses the Replace Action, the Action leads to Step 4. In this example, replacement requires approval, so the dashed Action transfers the task to an Inactive State, pushing the current Step into the Step Stack. At that point, the other Action leading from Step 4 is not enabled, due to an enabling condition which depends on receiving the approval. Once the approval arrives, the next Action pops the Step Stack to return to Step 4.

Note: a more complete scenario would probably also include Action that should lead from Step 5, for handling the case when approval is not granted, possibly leading to another State in the Closed category which reflects cancellation of the Work Request.

Figure 11 contains an example Task composed of three Activities, of which the state model of “Activity 1” is discussed in more detail:

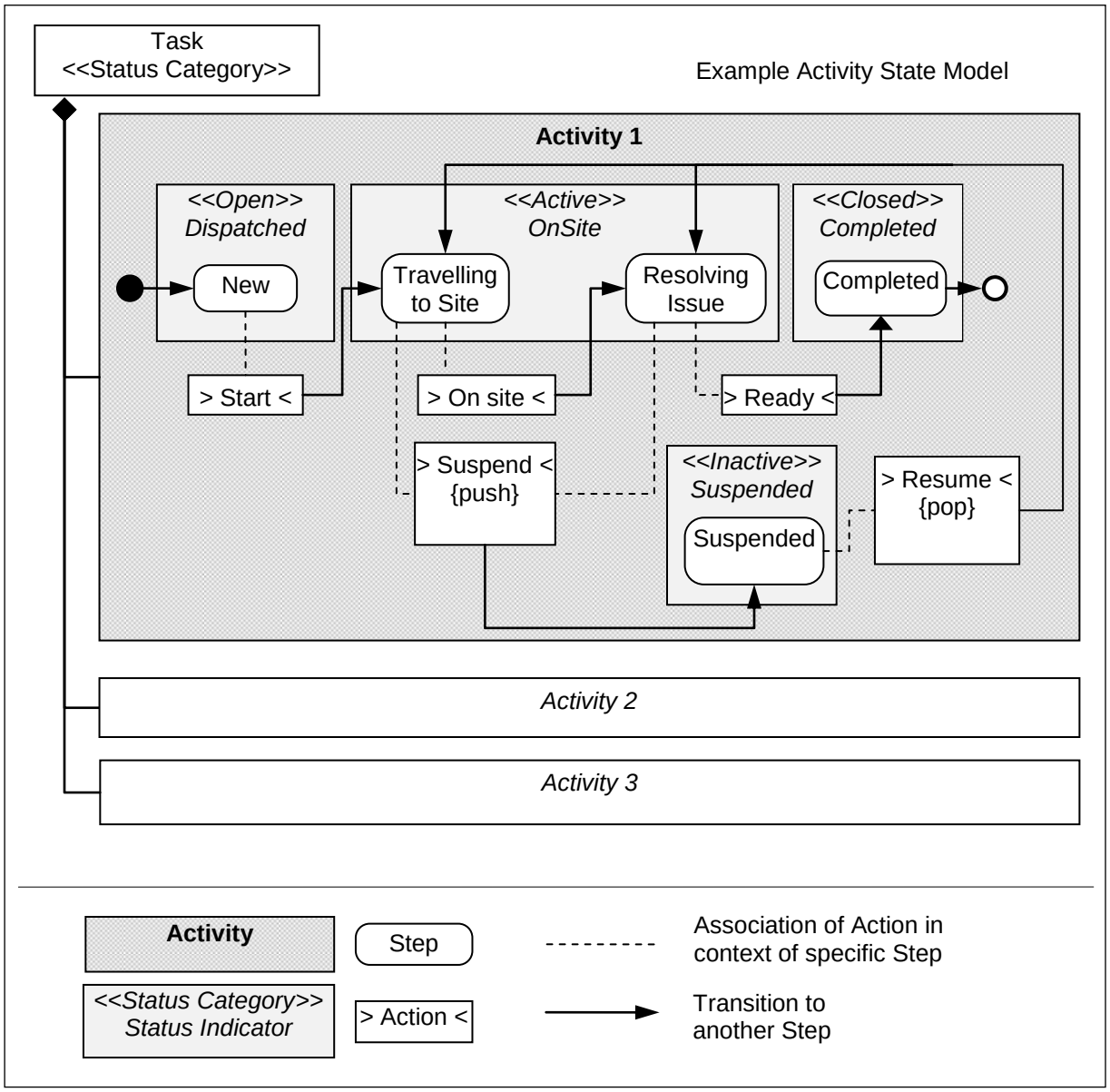


Figure 11: Activity State Model

In Figure 11, Activity 1 is composed of five Steps associated with four different States. Of these, Step "New" is the initial Step. Each State is associated with one Status Category, describing the overall status of the work while in that State.

Each Step, except for "Completed" has at least one Action associated. An Action, once invoked, transits the Activity to the designated target Step and the associated State.

While at the "Traveling to Site" and "Resolving Issue" Step, there are two Actions declared as the possible exit paths. As there are no Conditions associated with any of the Actions, the Implementation MUST offer choice of respective two Actions to the Assignee at these Steps.

The "Suspended" Step offers two exit paths through the same "Resume" Action, returning back to the Step where the "Suspend" Action was originally triggered. This is enabled by a concept of a *Step Stack* which MUST be supported by Implementation. An Action specifying a "{push}" classifier causes the identity of the Step triggering the Action to be stored into the top of the Step Stack. Consequently, an Action specifying a "{pop}" classifier causes the identity of the target Step to be retrieved from the top of the Step Stack rather than exactly identified in the state model. Each Activity has a separate Step Stack.

Status Categories and Status Indicators

As the creator of an Activity State Model, the Manager implicitly knows the semantic meaning of each State. Progress and status of the Activity is tracked based on information about State and Step transitions. However, the Implementation also needs some information about the semantic meaning of each State to be able to properly indicate and visualize the current status of the Activity to the Assignee. This information is conveyed using *Status Categories* and *Status Indicators*.

Each State MUST be associated with exactly one of the predefined Status Categories. The Status Category indicates the overall status of the Activity when it is in this State.

The following Status Categories shall be used.

- Open (i.e. Assignee has not yet started working on the Activity)
- Active (i.e. Assignee is actively pursuing the work in question)
- Inactive (i.e. Assignee has started work in question but has suspended it)
- Closed (i.e. Activity has been completed or it is of no relevance to Assignee anymore)

Additionally, each State MAY be also associated with one of the pre-defined or Implementation specific Status Indicators belonging to the Status Category associated with the State. The Status Indicator provides more fine grained information about the status of the Activity when it is in a particular State. Table 1 specifies pre-defined Status Indicators and their semantics. These MAY be used in Activity State Models. Implementation specific Status Indicators MUST begin with "X-".

Status Category	Status Indicator	Description
Open	Open	The initial default status. Indicates that the WR or Activity exists but is not currently assigned.
	Scheduled	The WR or Activity is scheduled, meaning that it has been assigned to specific Assignee(s) at specific times. It is not necessarily available to the Assignee (since it may be far in the future, or there may be a significant probability that new information will cause it to be assigned), but Assignees who query for WRs and Activities may still see items in Scheduled status (or any other status) subject to access rules imposed by Manager.

Status Category	Status Indicator	Description
	Tentative	The Activity is scheduled and assigned to specific Assignee(s) at specific time, and is available to the Assignee even though there remains some time before the actual dispatch, so the Manager may still re-assign the Activity. Rationale: In many cases, Assignees continuously need to have a glimpse into the plan for the rest of the day (or any other reasonably-close future period), even knowing that the plan is subject to change.
	Dispatched	The Activity is firmly assigned to specific Assignee(s), so that under non-exceptional circumstances (e.g. emergency), the Manager will not re-assign it to someone else.
	Acknowledged	The Assignee acknowledged the Activity assigned to him or her.
Active	EnRoute	Assignee is traveling towards service site
	OnSite	Assignee is on site
Inactive	Suspended	Work is on hold, pending some action such as delivery of parts
Closed	Rejected	The Assignee rejected the Activity assigned to him or her.
	Cancelled	The WR or Activity has been cancelled. This is typically an end-state that will not transition to any other State.
	Completed	The WR or Activity has been completed. This is typically an end-state that will not transition to any other State.
	Incomplete	The WR or Activity has been closed but not all the required work has been completed. Any further work will need to open a new WR or Activity. This is typically an end-state that will not transition to any other State.

Table 1: Pre-defined Status Indicators

5.1.2.5 Activity dependencies

Activities MAY have dependencies on other Activities being in specific States.

FFMII interface does not provide dedicated data constructs for modeling Activity and Action dependencies. Instead, both types of dependencies are modeled using Boolean expressions referred to as Conditions.

Activity-Enabling dependencies are specified as an Enable Condition associated with the Activity. If an Enable Condition is specified, the Implementation MUST NOT makes the Activity available to the Assignee, unless the Enable Condition evaluates to True. Once an Activity is initially made available to the Assignee, it MUST remain available regardless of the value of the associated Enable Condition.

Action-Enabling dependencies are specified as an Enable Condition associated with the Action. If an Enable Condition is specified, the Implementation MUST NOT allows triggering of the Action unless the Enable Condition evaluates to True.

The use of generic Boolean expressions makes it possible for Activities and Actions to be dependent also on other information, such as Work Request data values or system supplied values, including more complex expressions build on top of those. Conditions are in more detail discussed in Section 5.1.4.

Figure 12 extends the Activity State Model introduced in Section 5.1.2.4 by adding sample Activity dependencies:

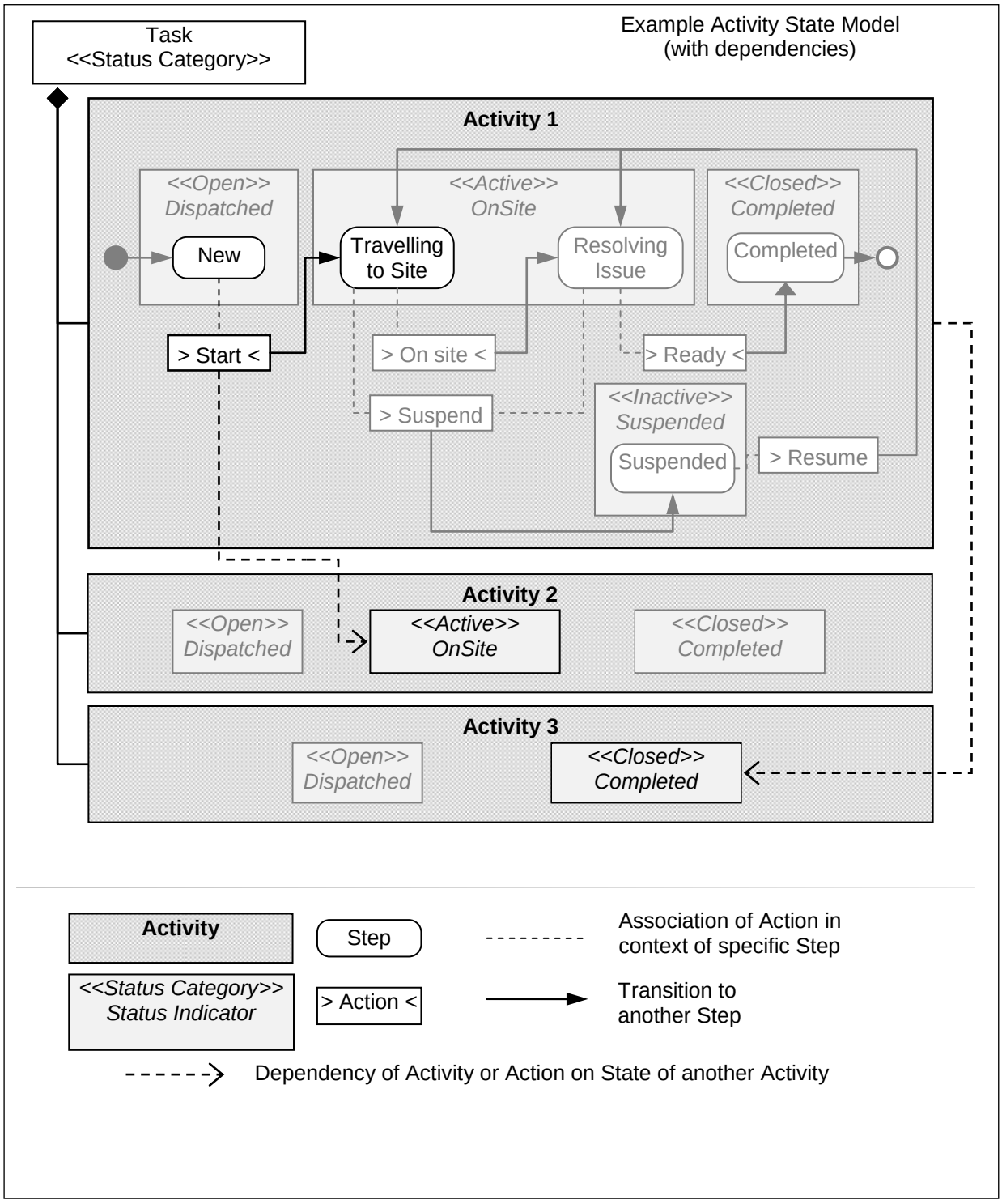


Figure 12: Activity State Model (Dependency)

In the example of Figure 12, Activity 1 is not made available to the Assignee until Activity 3 is in "Completed" State. Additionally, while at the "New" Step, Activity 1 won't be allowed to proceed towards the next Step, "Traveling to Site", unless Activity 2 is at any Step associated with the State "Ongoing". However, should Activity 2 proceed to the State "Completed" while Activity 1 is still in Step "New", the associated Action "Start" becomes disabled again. Activity 2 and Activity 3 have no dependencies on each other or Activity 1 and may therefore proceed independently at any time.

No explicit deadlock prevention

An improperly specified Activity State Models may result into deadlock on Task Level, i.e. a situation where the Task is not able to reach a Closed Work Request Status (see Section 5.1.5) by actions taken by the Assignee. Such situation may occur, for example, when there is a dependency loop between two Activities.

The Interface itself does not include proactive means of deadlock prevention. Instead, the Manager SHOULD ensure that each produced Work Request allows reaching its desired final Closed Work Request Status. Additionally, the Manager MAY cancel any Work Request using WR_INVOKE_ACTION operation as described in Section 8.2.2.

5.1.2.6 Work Type Specification change constraints

Work Type Specifications stored in the "WorkTypes" repository MAY be updated at any time via Reference Data Management. However, the Implementation MUST retain internally and apply to a Work Request the version of the Work Type Specification that was used at the time the Work Request was initially created or last updated. Therefore, from Work Request perspective the associated Work Type Specification may only change when the Work Request itself is updated.

When an existing Work Request is updated, the associated Work Type Specification MUST NOT changes in a way that would void integrity of the existing data in the associated Work Request Status Record or otherwise contradict the current Work Request State. If the constraints are violated then the Implementation MUST reject Work Request update with error code E3017 ILLEGAL_WTS_UPDATE (See Section 8.7). The following changes constraints MUST be honored.

- Existing Activities MUST NOT be removed.
- If Work Request Status (see section 5.1.5) is Closed before the update then new Activities MUST NOT be added.
- The new state model associated with existing Activities MUST contain the current Step of the Activity.
- Status Category of State associated with the current Step of each existing Activity MUST NOT change.

5.1.3 Schedule

An Activity MUST be constrained by a Schedule.

The Schedule associated with an Activity has two logical parts: The **time constraints** defining when the Activity may be executed; and the **planned time** for executing the Activity;

The Schedule MUST have a "**time constraints**" part, which has the following attributes:

Latest Start (Mandatory)

A date-time data element, specifying the latest time when the Activity may be started. Note that this does not constrain the time when the Activity may be finished.

Earliest Start (optional)

A date-time data element specifying the earliest time when the Activity may be started (if not specified, it is assumed that the Activity may be started at any time between the present and the Latest Start).

Latest Finish (optional)

A date-time data element specifying the latest time when the Activity may be finished (if not specified, the finishing time is not constrained).

Appointment Start, Appointment Finish (optional)

Date-time data elements specify the start and end of the appointment that is the time window during which the service provider has promised that the service would be delivered. If one of these elements is specified, the other one MUST be specified as well.

The Appointment constraint consists of Appointment Start and Appointment Finish. If specified, Appointment constraint supersedes other constraints. It is intended for use in cases similar to the following common use case: Assume the customer has a service contract specifying that service must be provided within 2 business days from the time when the customer requested the service. Therefore, if the customer calls on Tuesday noon, the Latest Start will be set to Thursday noon. During the interaction between the customer and the service provider, they may set a service appointment for Wednesday between 10AM and noon. This will be specified in the Appointment Start and Appointment Finish elements. Still, it is useful for the Assignee to know the Latest Start data: for example, the Assignee might have some delays and check which of the Assignee's planned Activities may be moved and still meet the original Latest Start.

The Activity MAY have a "**planned time**" part. If this part exists, it MUST include a "**Planned Start**" date-time data element. In the above example, where the service appointment was set for Wednesday between 10AM and noon, the Planned Start element might show that the Activity was set to start at 11AM. It MAY also include a "**Planned Finish**" date-time data element.

No enforcement of logical relationships between the Schedule elements: The Manager and Implementation MAY enforce logical relationships between the Schedule elements. For example, it is permissible for the Planned Start to be in violation of the timing constraint specified by the Latest Start element. Such a situation may arise, for example, if the Manager is unable to schedule the Activity in such a way that obeys the constraint (possibly due to lack of work capacity) but schedules it at a later time, since being late may be better than not performing the Activity at all.

5.1.4 Data Forms

5.1.4.1 Overview

Data Forms are used to model dynamically specified structured information. Data Forms are used, for example, for the purpose of defining Work Request header, overview and instructions, Step level instructions and user input.

Data Forms related to Work Requests are declared using Data Forms as part of Work Type Specification. Data Forms related to Field-Initiated Requests (Request Form, ResponseForm) are declared using Data Forms as parts of Field-Initiated Request Specification.

A Data Form (DataForm) MUST contain one or more Data Elements providing descriptive information about the associated values (see sections 5.1.4.2 and 5.1.4.3 for details). A Data Element is either a Data Element Specification or in case of Data Form declared by a Work Type Specification, a reference to share a Data Element Specification specified by the Work Type Specification. This allows the same Data Element Specification to be used as part of several Data Forms defined by the same Work Type Specification.

The actual values of Data Elements are provided by a Work Request or a Field-Initiated Request associated with the specification declaring the data Form. Data Values are provided as a set of Data Bindings. Data Bindings are also used when referring to information provided by the Assignee.

Figure 13 illustrates the structure of a Data Form and its relations to the declaring specification as well as to the Work Request or Field-Initiated Request instance providing the actual data values.

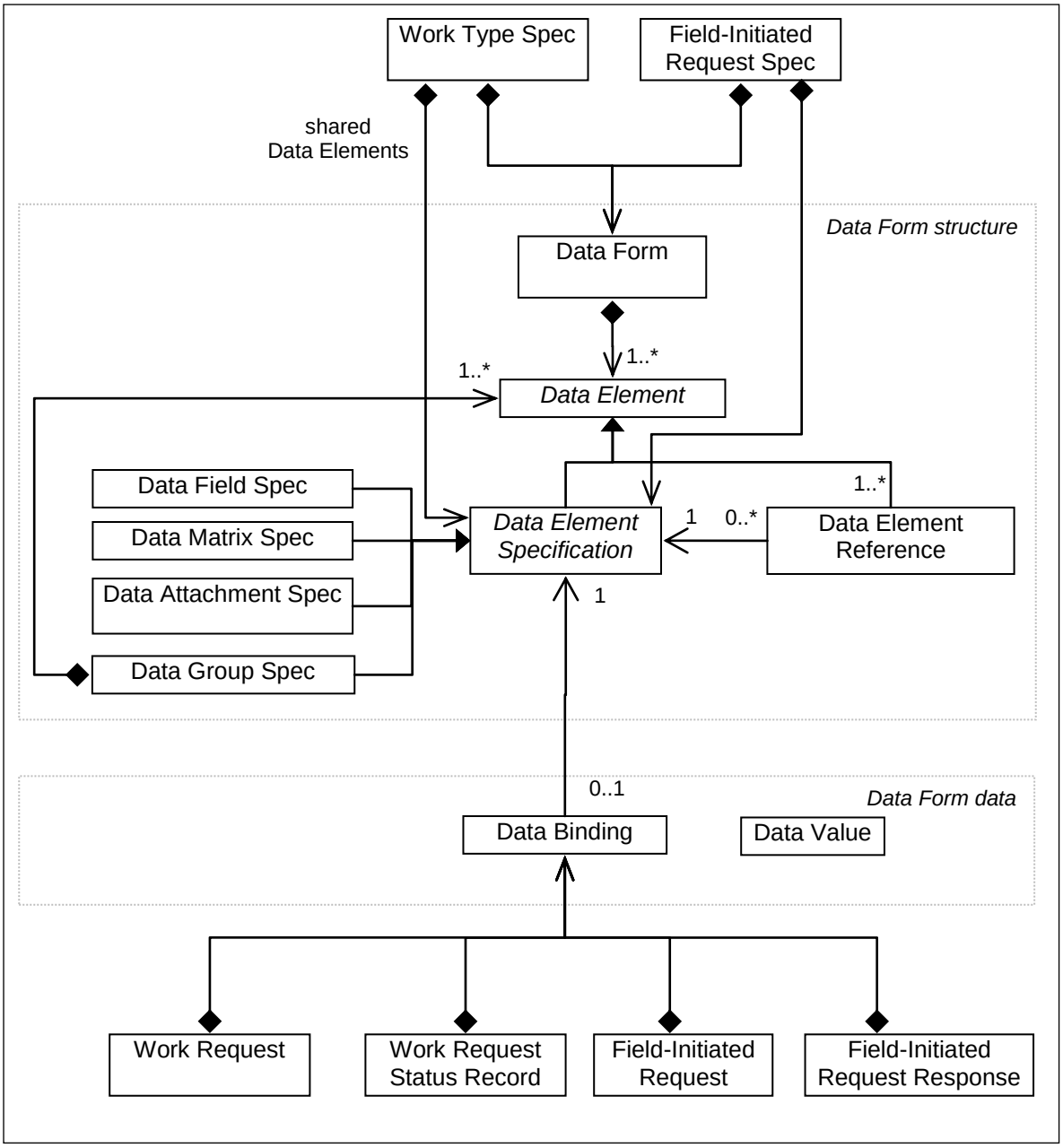


Figure 13: Data Form Data Types

5.1.4.2 Data Element Specification (abstract)

Data Element Specification itself is an abstraction that supports a common set of attributes as per the following figure:

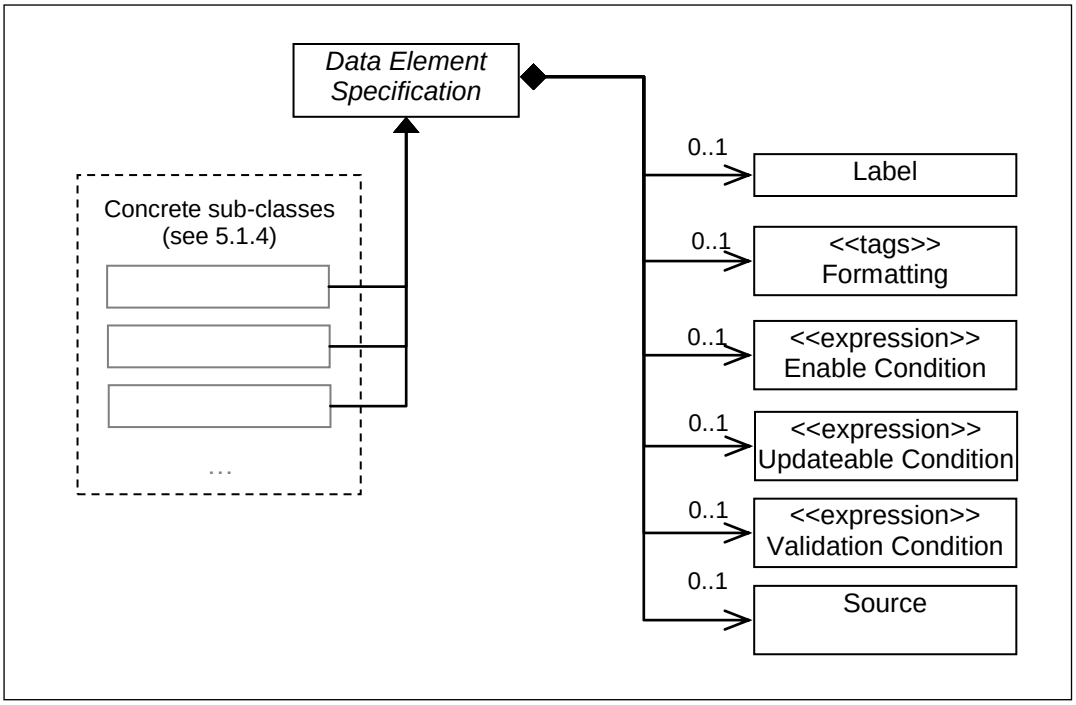


Figure 14: Data Element Specification

Label [Optional]

Specifies text identifying the associated Data Element on the user interface.

Formatting Tags [Optional]

Specifies the way the associated Data Element SHOULD be visualized using a sequence of standard or custom formatting tags. Standard tags are specified in Section 8.5.6. Additionally, an Implementation MAY introduce own set of custom tags is necessary. Names of Implementation-specific tags MUST" be prefixed with "X-".

Enable Condition [Optional]

An expression that specifies when the associated Data Element is in enabled state. When NOT enabled, the Data Element MUST not be shown to the user, and its content MUST not be validated.

Default value of Enable Condition is True (i.e. Data Element having no Enable Condition defined is regarded as enabled).

Updateable Condition [Optional]

An expression that specifies when the content of the associated Data Element can be updated by the user. In order to be updateable, the Data Element in question MUST also be enabled.

Default value of Updateable Condition is False (i.e. Data Element having no Updateable Condition defined is regarded as non-updateable) in case of Header, Work Overview, Work Instructions and Activity Location Data, Data Forms. Default value of Updateable Condition is True in case of an Action Input Form.

Validation Condition [Optional]

An expression that specifies an expression used to validate user input of updateable Data Elements.

Default value of Validation Condition is True (i.e. value of Data Element without Validation Condition is not checked for validity).

Source [Optional]
Identifies the expected source of Assignee provided data and provides a hint on how the Implementation SHOULD obtains the data, such as using a camera. See Section 8.5.7 for a list of standard source identifiers.

Enable Condition, *Updateable Condition* and *Validation Condition* are conditions specified as Expressions. These Expressions evaluate into Boolean True or False. Expressions refer to Work Request Data Elements, system properties, elements of Reference Data or defined constants, and combinations thereof. Expressions are in more detail discussed in Section 8.6.

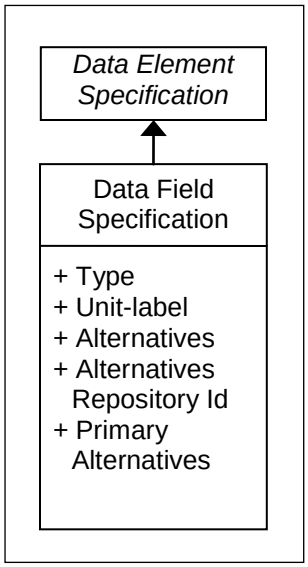
5.1.4.3 Data Element Types

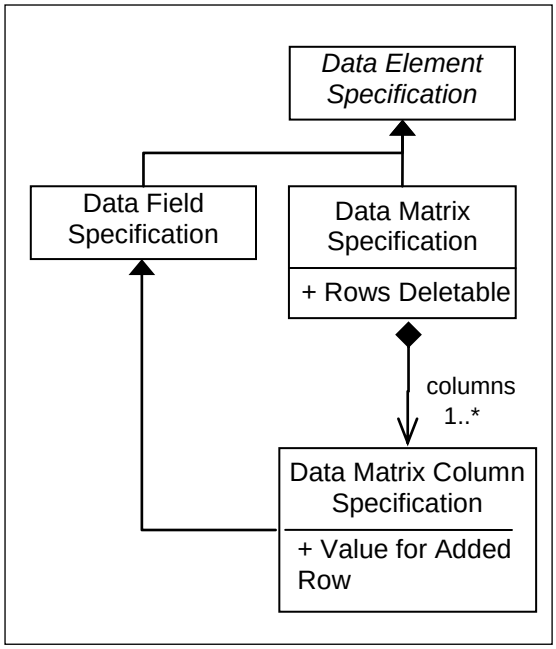
Types of concrete Data Element Specifications are:

Data Field Specification
Specifies a data field displaying or accepting a single value.

A Data Field Specification MUST specify the type of the associated value (one of primitive data types such as String, Integer or Boolean, see Section 7.2) and it MAY specify a unit label to be displayed along the value. A Data Field Specification MAY specify the set of valid value alternatives, effectively resulting into a multiple choice field. The value alternatives are specified either directly or by referring to Reference Data. A Data Field Specification MAY also specify one or more primary alternatives. Primary alternatives specify the values an Assignee is most likely to provide as input data on an updateable data field and the Implementation MAY leverage those, for example to improve user experience.

Data Field Specification is described in more detail in Section 8.5.





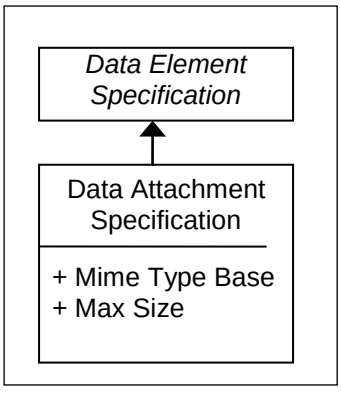
Data Matrix Specification

Specifies a two-dimensional matrix of data composed of rows and columns.

Data Matrix Specification MUST declares one or more columns using Data Matrix Column Specification which is a kind of Data Field Specification. The attributes of Data Field Specification, such as label and type, are applied to the column and the values contained in the column. Data Matrix Column Specification MAY also specify a default value to be used automatically in the corresponding column for any new row added by the Assignee.

Data Matrix Specification MAY also specify whether rows of an updateable matrix can be deleted by the Assignee. By default the rows of an updateable matrix can be deleted.

Data Matrix Specification is described in more detail in Section 8.5.4.

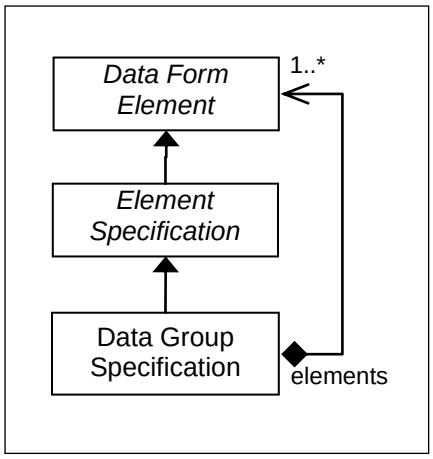


Data Attachment Specification

Specifies an unstructured data object, such as an image or a document, to be made available to or provided by the Assignee.

When used for user input (i.e. an updateable element), Data Attachment Specification MAY specify the expected base MIME type from [RFC2046] (e.g. “image”) and the maximum

allowed size for attachment data provided by the Assignee.
Data Attachment Specification is described in more detail in Section 8.5.3.



Data Group Specification

Specifies a group of other Data Elements.

A Data Group Specification MUST contains one or more other Data Elements declared as Data Element Specifications. Data Group Specifications MAY be nested within each other. The Implementation MUST regard the contained elements as pieces of related information and SHOULD visualize those in such a way that grouping is visible to the user.

Data Group Specification MAY introduce own Enable Condition, Updateable Condition or Validation Condition that have a cascading effect on the contained elements.

Data Group Specification is described in more detail in Section 8.5.5.

5.1.5 Work Request Status and Work Request Administrative Closing Status

While each Activity has its own state model, from the Manager point of view it is also important to be able to determine the overall status of a Work Request. For example, in order to save network bandwidth, a Manager MAY want to ignore any Work Request that is already in a closed State or, on the other hand, query the status of only those Work Requests that have not yet been started.

The Implementation MUST maintain Work Request Status for each Work Request. The current Work Request Status of a Work Request MUST also be made available to the Manager as part of the associated Work Request Status Record (see Section 5.1.6 for details).

Work Request Status is implicit, and its value is based on the Status Category associated with the current State of each included Activity and weather the Administrative Closing Status has been asserted or not. Work Request Status is indicated using the same Status Categories (See Table 1) as for indicating the overall status of an Activity (see Section 5.1.2.4).

The following diagram depicts Work Request Status and its relations to other data types.

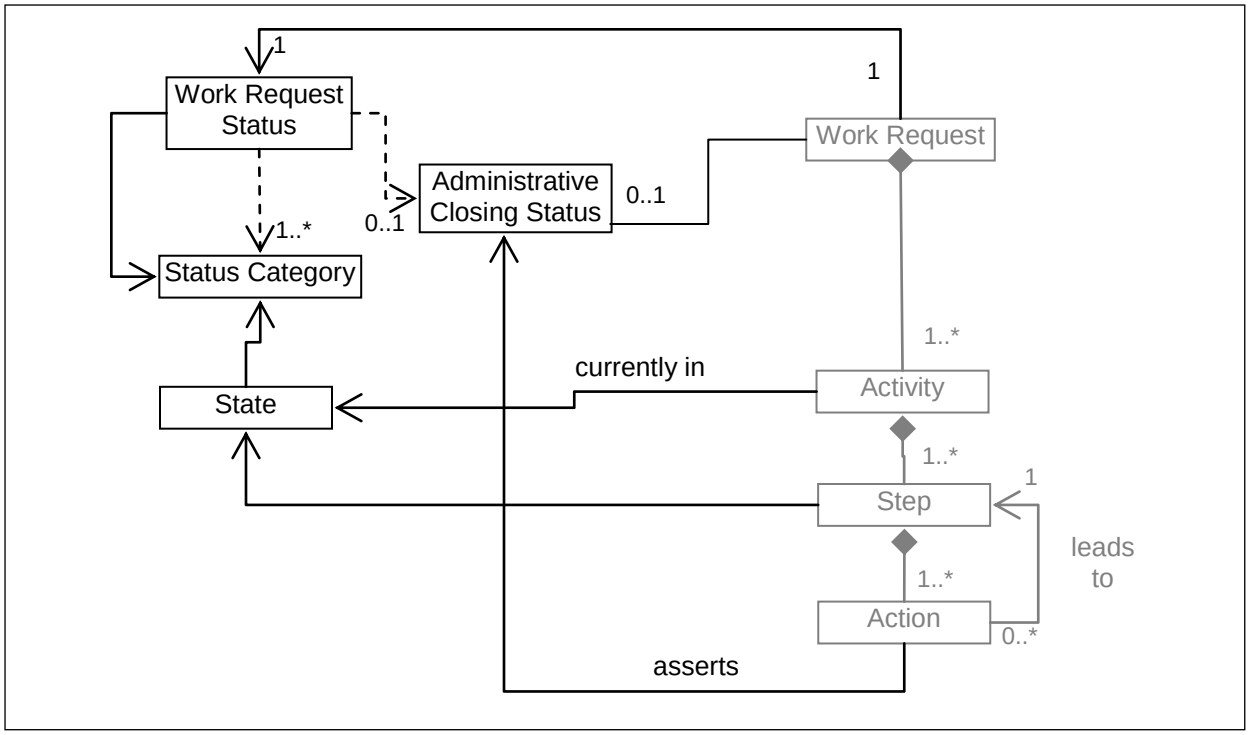


Figure 15: Work Request Status and Task Relationship

Work Request Status changes when Actions are performed on Activities of the Task changing the current State of the Activity.

An Action MAY also assert the Administrative Closing Status to force the Task Status to “Closed”, regardless of the current State of each included Activity. The Action MAY assert the Administrative Closing Status to any value. The chosen value itself does not affect Task Status; however the Manager MAY use different values to keep track of the reason the Task was closed. The asserted value of the Administrative Closing Status, if any, is exposed to Manager as part of the Work Request Status Record (see Section 5.1.6).

The following chart depicts the possible Work Request Status values and the possible transitions between them.

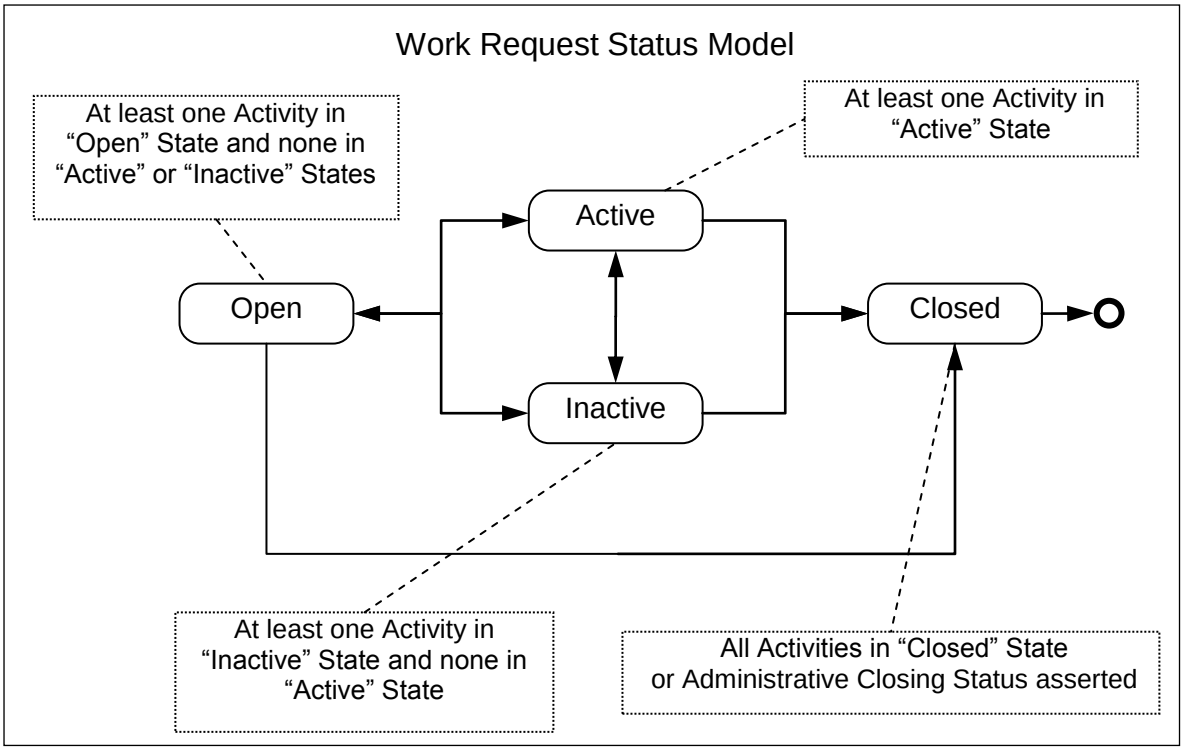


Figure 16: Work Request Status State Model

Work Request Status is determined by the following set of rules. The rules are evaluated in the listed order and the first rule that can be applied determines Work Request Status.

- Work Request Status is "Closed" if all included Activities are currently in a State associated with Status Category "Closed" or if Administrative Closing Status has been asserted.
- Otherwise, Work Request Status is "Active" if at least one included Activity is currently in a State associated with Status Category "Active". No Administrative Closing Status has been asserted.
- Otherwise, Work Request Status is "Inactive" if at least one included Activity is currently in a State associated with Status Category "Inactive". No included Activity is currently in a State associated with Status Category "Active" and no Administrative Closing Status has been asserted.
- Otherwise, Work Request Status is "Open". At least one included Activity is currently in a State associated with Status Category "Open" and none in a State associated with Status Categories "Active" or "Inactive" and no Administrative Closing Status has been asserted.

Once Work Request Status becomes "Closed" the Implementation MUST prevent further Actions from being performed on any of the included Activities, making this the last transition of the Task state model.

5.1.6 Work Request Status Record

Work Request Status Record is a data structure reflecting state changes of Work Request after it has been received by the Implementation. An Implementation MUST maintain one Work Request Status Record per each Work Request, and make it available for retrieval through appropriate interface operations.

Work Request Status Record MUST contain exactly one *Tasks Status Record*, and zero or more *Change History Entries* as indicated in Figure 17:

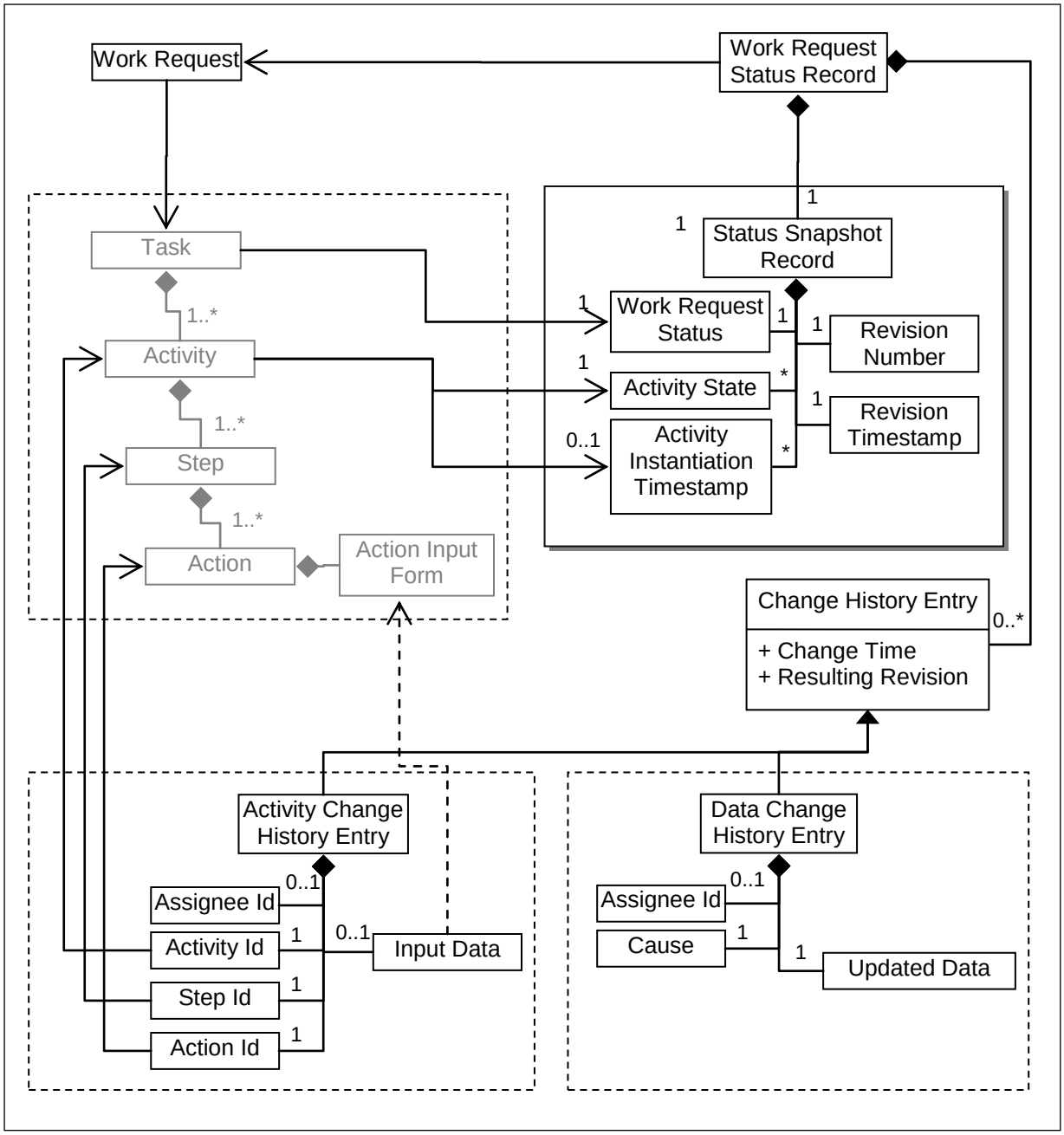


Figure 17: Work Request Work Request Status Record

A Status Snapshot Record is a snapshot of current Work Request status information. A Status Snapshot Record MUST contains the following data elements unless indicated as optional:

- Current **Work Request Status** determined as described in Section 5.1.5
- Whether **Administrative Closing Status** has been asserted and its value, if any
- Current **Activity State** of each defined Activity
- An **Activity Instantiation Timestamp** indicating the time when the Activity was made available to the Assignee (Optional, present if and only if made available to the Assignee)
- A monotonically increasing **Revision Number** reflecting the number of changes made to the Work Request by Assignee or Manager since the creation of the Work Request
- **Revision Timestamp** for the latest revision

Change History Entries record the circumstances, under which an Activity transited from one Step to another (Activity Change History Entry) as well as Work Request data changes made by Assignees (Data Change History Entry). Each Change History Entry MUST contains the following common data elements:

- **Timestamp** of the change
- Resulting **revision number** of the Status Snapshot Record

Activity Change History Entries record Activity State Model transitions initiated either by an Assignee or a Manager invoking an Action on the Activity. Activity Change History Entries MUST contains the common data elements for Change History Entries and MUST additionally include the following data elements unless indicated as optional.

- Identification of the **Assignee** that initiated the transition,(Optional, present only in Assignee initiated transitions)
- An identification of the related **Activity**
- An identification of the **Step** and the associated **State** the Activity reached after the transition
- An identification of the **Action** that triggered the transition
- Any **input data** the Assignee supplied in connection to the Action. (Optional present only if input data was supplied)

Data Change History Entries record Work Request data changes caused by direct manipulation of updateable Data Elements by the Assignee or by Data Update Operations (specified in Section 8.2.1.5) associated with Actions invoked by an Assignee or the Manager. Direct Work Request data updates by Manager are not recorded in Work Request Status Record. Data Change History Entries MUST contain the common data elements for *Change History Entries and* additionally the following data elements unless indicated as optional.

- Identification of the **Assignee** that updated Work Request data ,(Optional, present only in Assignee initiated transitions)
- Identification of the **cause**, whether caused by an Action or direct manipulation of updateable Data Element
- Any **updated data**

5.2 Reference Data

An Implementation MAY provide means for the Manager to establish custom data repositories with arbitrary content. Content of such repositories is commonly denoted as “Reference Data”, and MAY be used for input value selection, lookup of display values or content validation in Work Requests.

An Implementation MAY also provide access to system repositories providing access to selected data on Implementation side, such as Assignee identities. System repositories have reserved identifiers, and they MAY impose restrictions on their content.

Figure 18 presents the domain model of Reference Data in FFMII context:

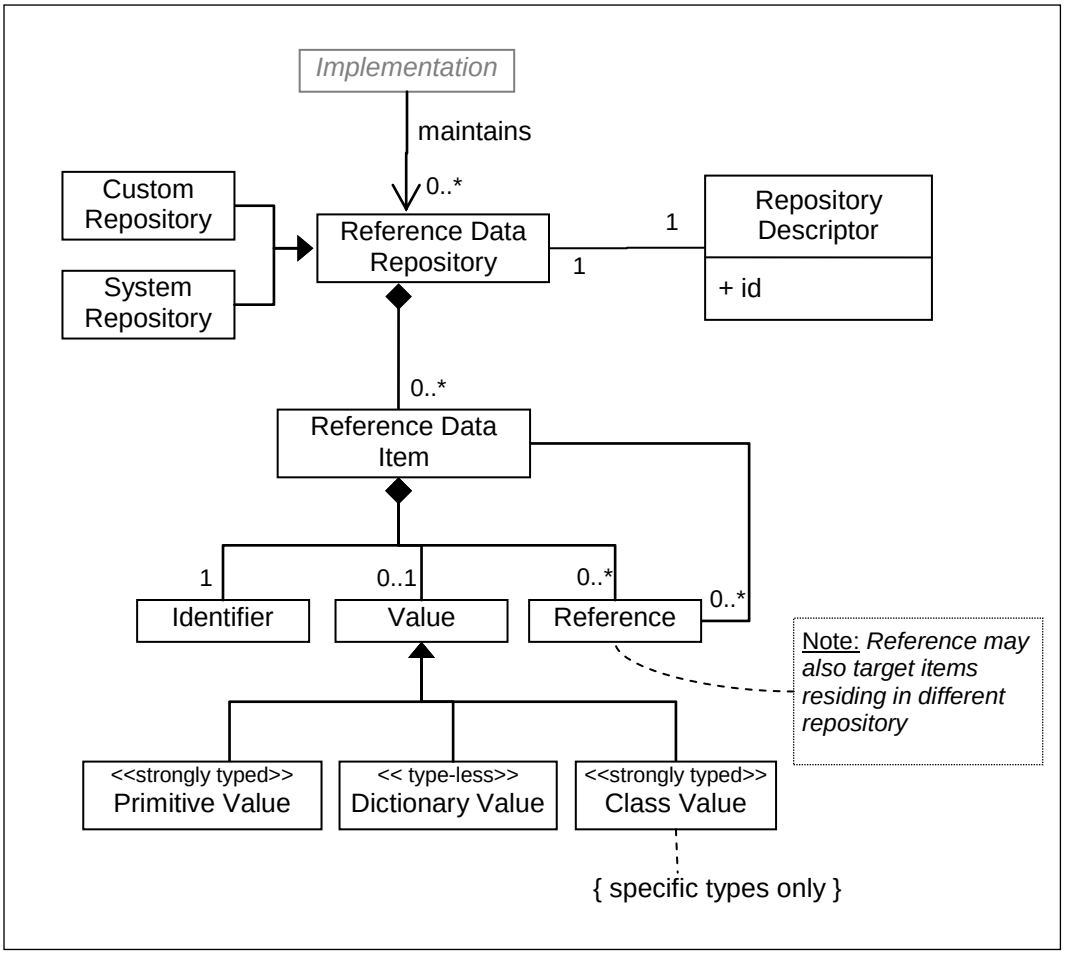


Figure 18: Reference Data Overview

Each Reference Data Item stored in a Reference Data Repository MUST have an identifier and MAY have a value. The value MUST be either a primitive value, a dictionary value, or a strongly-typed class value matching the repository constraints.

Each Reference Data Item MAY also have zero or more references to other Reference Data Items residing in the same or different Reference Data Repository managed by the same Implementation. The semantics of the references depends on the type of the repository and the context in which the data is used. A single set of operations may be used to manage the references.

Reference Data Management is described in detail in section 8.3.

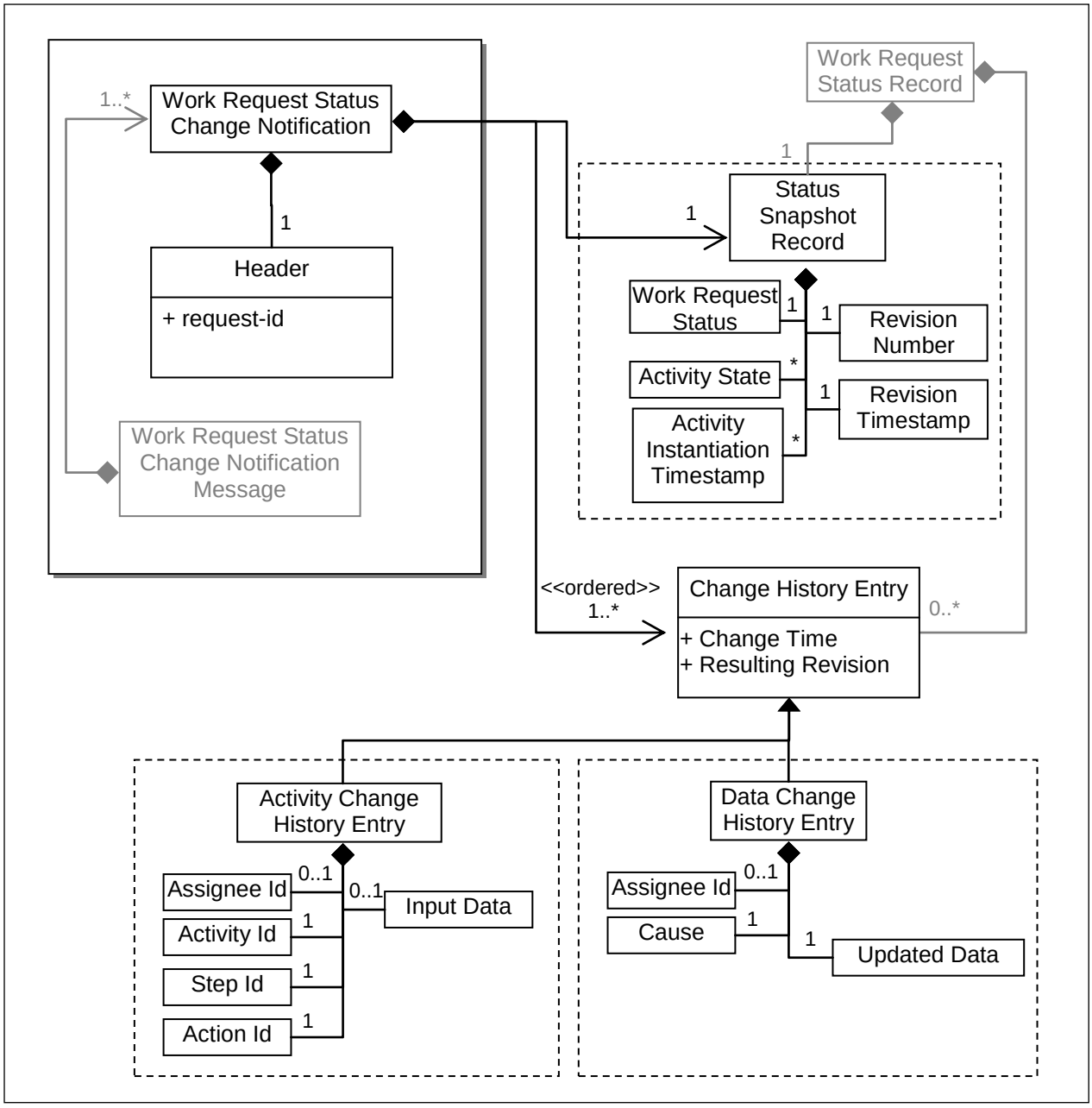
5.3 Work Request Status Change Notification

A *Work Request Status Change Notification* informs Manager of changes to a Work Request it has deployed to the Implementation. This off-loads Manager from having to poll for status updates frequently.

A *Work Request Status Change Notification* contains a copy of the *Tasks Status Record,* and one or more *Change History Entries* (see Section 5.1.6) covering data and Activity changes of the Work Request since last successful notification. Each transition in the Activity State Model MUST be notified as an Activity Change History Entry. Any change on Data Form content MUST be notified as a Data Change History Entry. In addition, a Header element is provided to identify the Work Request in question.

A single message may contain several *Work Request Status Change Notifications* concerning several Work Requests if the Manager has advertised the corresponding capability (see Section 8.1).

Figure 19 provides an overview of Work Request Status Change Notification structure:



690

691 **Figure 19: Work Request Status Change Notification**

692 **Feature availability**

693 The ability to dispatch and receive Work Request Status Change Notifications is optional for
694 Implementation and Manager, respectively. If supported, Implementation and Manager MUST indicate
695 availability of the feature through appropriate Capability Descriptor as described in Section 8.1 of Identity
696 and Capabilities Discovery chapter.

697 **Bundling of Activity History Entries**

698 An Implementation MAY combine several Change History Entries within a single Work Request Status
699 Change Notification. Reason for such behavior might be re-transmissions caused by transient network
700 errors, asynchronous dispatch of notifications, or matter of policy for saving network bandwidth. The
701 Implementation MUST preserve chronological order of Change History Entries included in a single Work
702 Request Status Notification.

703 **Chronological order of notifications**

The Implementation MUST always send Work Request Status Change Notifications associated with a specific Work Request in chronological order. A Work Request Status Change Notification MUST NOT be sent if some earlier notification associated with the same Work Request has not been sent or has not been successfully acknowledged by the Manager.

However, the Implementation SHOULD continue sending Work Request Status Change Notifications for other Work Requests to prevent a problem with a single Work Request from stopping the flow of all notifications. The Implementation SHOULD also retry delivery of failed notifications, subject to implementation-specific policy.

Manager-side constraints

Through Capability Descriptors, a Manager MAY specify constraints concerning processing of Work Request Status Change Notification messages. An Implementation MUST comply with specified constraints. The available Manager-side constraints are described as part of the corresponding Capability Descriptor in Section 8.1.3.2.1.

Specification of delivery target

Work Request Status Change Notifications logically target the Manager that created the corresponding Work Requests. Resolution of the delivery end-point (such as URL) is implementation-specific, and not subject to the FFMII interface specification.

5.4 Field-Initiated Request

5.4.1 Introduction

Field-Initiated Request (FIR) is a request initiated by an Assignee and dispatched as a structured message from Implementation to Manager. It is intended for making requests or reporting information outside the usual Activity work flow, such as requesting activation or reset of a specific device, reporting absence of the Assignee, or requesting additional work for the Assignee.

A Field-Initiated Request is either a *Topical Inquiry* or a *Topical Notification,* depending whether the Manager is expected to return data as a response or not, respectively.

Each Field-Initiated Request MUST address exactly one *Topic*. A Topic determines operation semantics on the Manager side and the required request data content. A Topic also defines whether the request should be associated with a Work Request or not. The Manager processes each request based on the addressed Topic and request content. This may result into consequent interaction between the Manager and another system integrated with it. Such interactions, however, are transparent to Implementation and outside the scope of the FFMII.

Topics are Manager specific and they are registered with the Implementation by storing a Field-Initiated Request Specification into repository “FieldInitiatedRequests” via Reference Data Management. See Section 5.2 for details on Reference Data Management and the Field-Initiated Requests repository.

Figure 20 presents a concept model of a Field-Initiated Request.

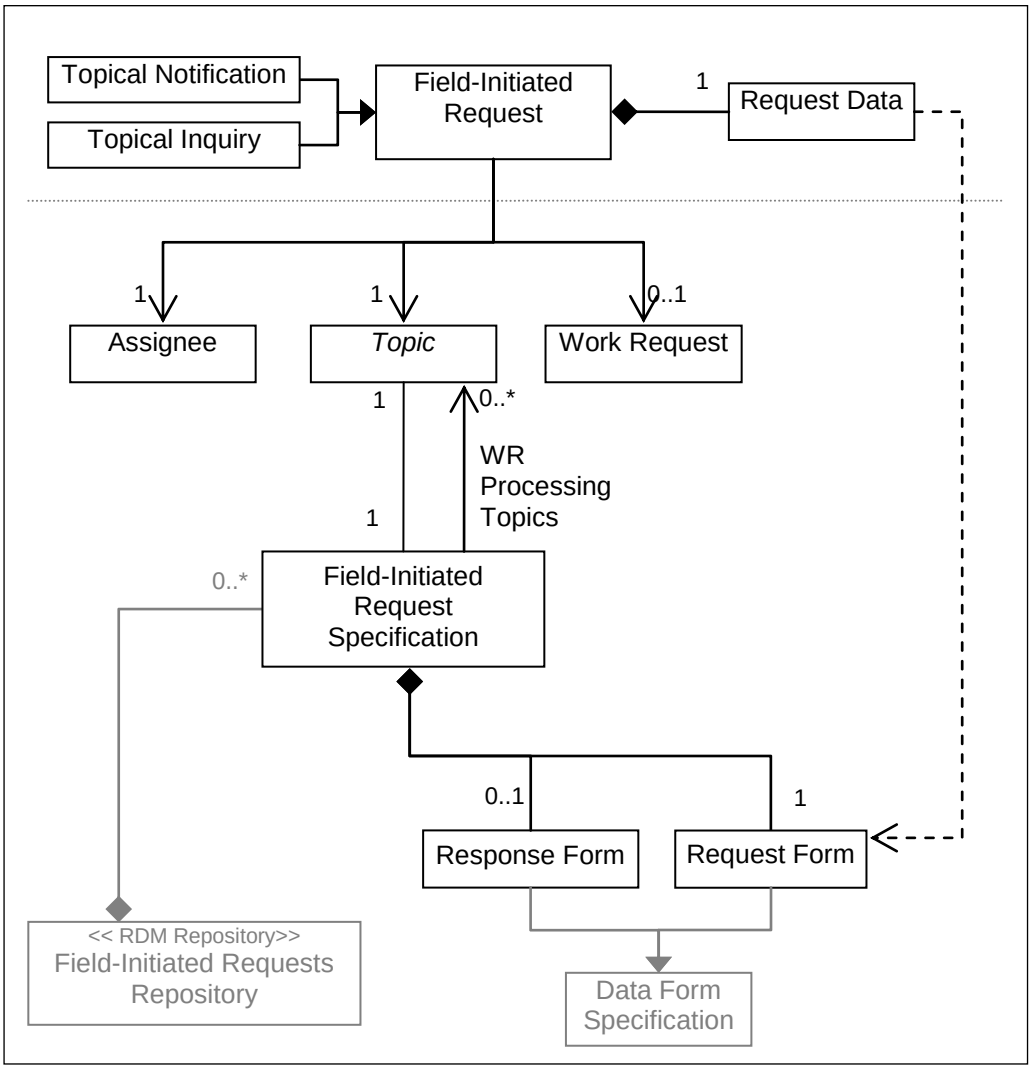


Figure 20: Field-Initiated Request

A Field-Initiated Request is a message transmitted from Implementation to Manager and it contains.

- Unique **Identifier** generated by the Implementation (Mandatory)
- Request initiation **Timestamp** (Mandatory)
- Identifier of the addressed **Topic** (Mandatory)
- Identifier or the initiating **Assignee** (Mandatory)
- Invocation context as **Work Request**, **Activity** and **Step** identifiers; mandatory if invoked from the context of a specific Work Request (Optional)
- **Request data** supplied by the Assignee (Mandatory, but may be empty if allowed by corresponding Request Form specification)

Request data included with a Field-Initiated Request MUST conform to the structure and Validation Condition defined by *Request Form* included as part of the Field-Initiated Request Specification associated with the Topic.

For each Field-Initiated Request there MUST be an existing Field-Initiated Request Specification with the same Topic identifier. The Field-Initiated Request Specification is provided dynamically by the Manager and stored into the Field-Initiated Request repository using Reference Data Management.

A Field-Initiated Request Specification contains following information.

- Identifier of the associated **Manager** (Mandatory)
- Identifier of the associated **Topic** (Mandatory)
- **FIRType** of the Topic, Topical Inquiry or Topical Notification (Mandatory)
- **Topic Label** to be used for visualization purposes by the Implementation (Mandatory)
- **Group Label** to be used for visualizing grouping of different Topics (Optional)
- **Request Form** specifies data to be supplied by the Assignee (Mandatory, but may be empty)
- **Response Form** specifies data to be returned, if FIRType is a Topical Inquiry (Optional, may be empty)
- Whether Manager **returns Work Requests** as a response, if FIRType is Topical Inquiry (Mandatory)
- Set of available **Work Request processing Topics**, if Manager returns Work Requests (Optional)
- Whether resulting FIR must be **bound to a Work Request** or not (Mandatory)

Request and response data is modeled as a Data Form Specification which is described in more detail in Section 5.1.4.

The Field-Initiated Request Specification indicates whether Manager returns Work Requests when FIRType is Topical Inquiry. Such inquiries can be used, for example, to request additional work for the Assignee. If Work Request processing Topics have also been specified then the Assignee should be able to choose any of the returned Work Requests and initiate Field-Initiated Requests associated with the listed Topics on any of the returned Work Requests.

Support for Field-Initiated Requests is optional. Manager MAY support incoming Field-Initiated Requests. Implementation MAY support the Field-Initiated Requests repository.

Field-Initiated Requests are described in more detail in Section 8.4.

5.4.2 Declaring available Field-Initiated Requests within a Work Type Specification

Field-Initiated Request is bound to a Work Request if (and only if) it is declared to be available in and invoked from within the work flow of the associated Work Type Specification.

For each Step within an Activity, a Work Type Specification MAY declare any number of available Topics for which Field-Initiated Requests may be initiated, as illustrated in Figure 21. Additionally, each available Topic MAY have an associated Enable Condition making the request available to the Assignee only if specified pre-requisites are met.

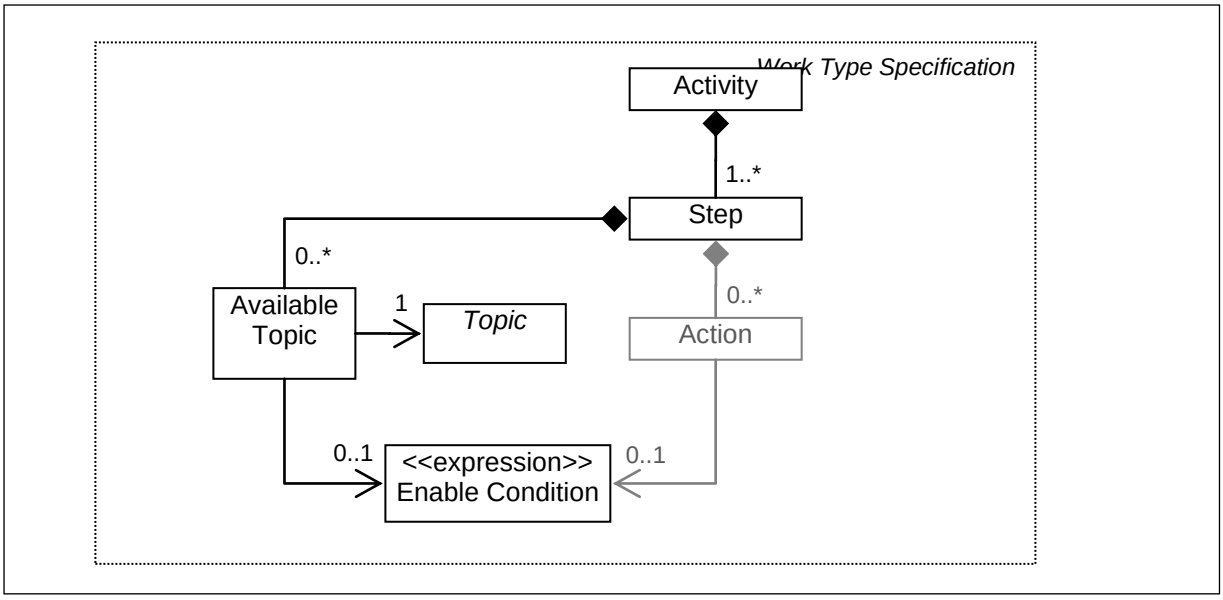


Figure 21: FIR within Work Type Specification

5.4.3 Responses to Topical Inquiries

A response to a Topical Inquiry is delivered from Manager to an Implementation as a structured response message, *Field-Initiated Request Response*. The response MUST refer to an earlier request and its information content MUST conform to the specification associated with the request.

The Manager MAY send any number of intermediate responses for a request before sending the final response containing the Final indicator having the value True. Consequent responses to the same request are identified and ordered by a monotonically increasing sequence number. An Implementation SHOULD regard Intermediate responses as progress indications and a later response MUST override any information contained in previous responses to the same request.

Figure 22 outlines the concept model for Field-Initiated Request Response.

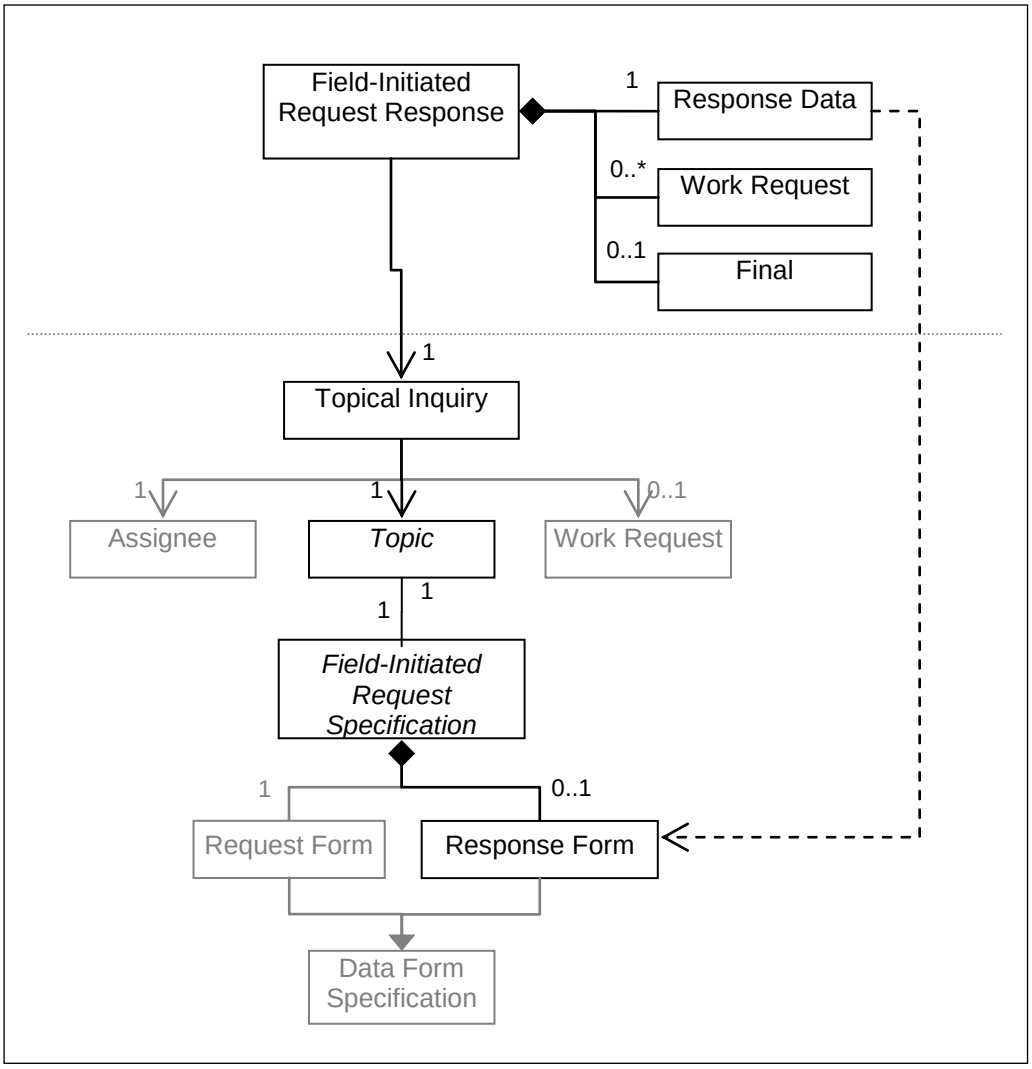


Figure 22: Field-Initiated Request Response

Field-Initiated Request Response contains following information.

- Identifier of the associated **Field-Initiated Request** (Mandatory)
- **Sequence Number** starting with 0 and monotonically increasing for further responses associated with the same Field-Initiated Request (Mandatory)
- Response **Timestamp** (Mandatory)
- **Response Data** returned by the Manager (Mandatory, but may be empty)
- Any number of **Work Requests** returned by the Manager, if allowed by the specification associated with the request (Optional)
- Whether response is **Final** or not (Mandatory)

Response data included with a Field-Initiated Request Response MUST conform to the structure defined by *Response Form* included as part of the Field-Initiated Request Specification associated with the Topic. Response MUST NOT include Work Requests unless specifically allowed by the associated Field-Initiated Request Specification.

6 FFMII Interface Architecture

6.1 Overview

The FFMII interface logically divides into several functional layers and operations domains. Out of these, some functionality is targeted at Implementation and some at Manager. Both Manager and Implementation MUST conform to the applicable parts the Core Data Model, implement functionality necessary for identity and capabilities discovery, and comply with the requirements of message handling, protocol binding and transport as prescribed by the Interface.

Figure 23 outlines subsystems of FFMII interface and their relative positioning:

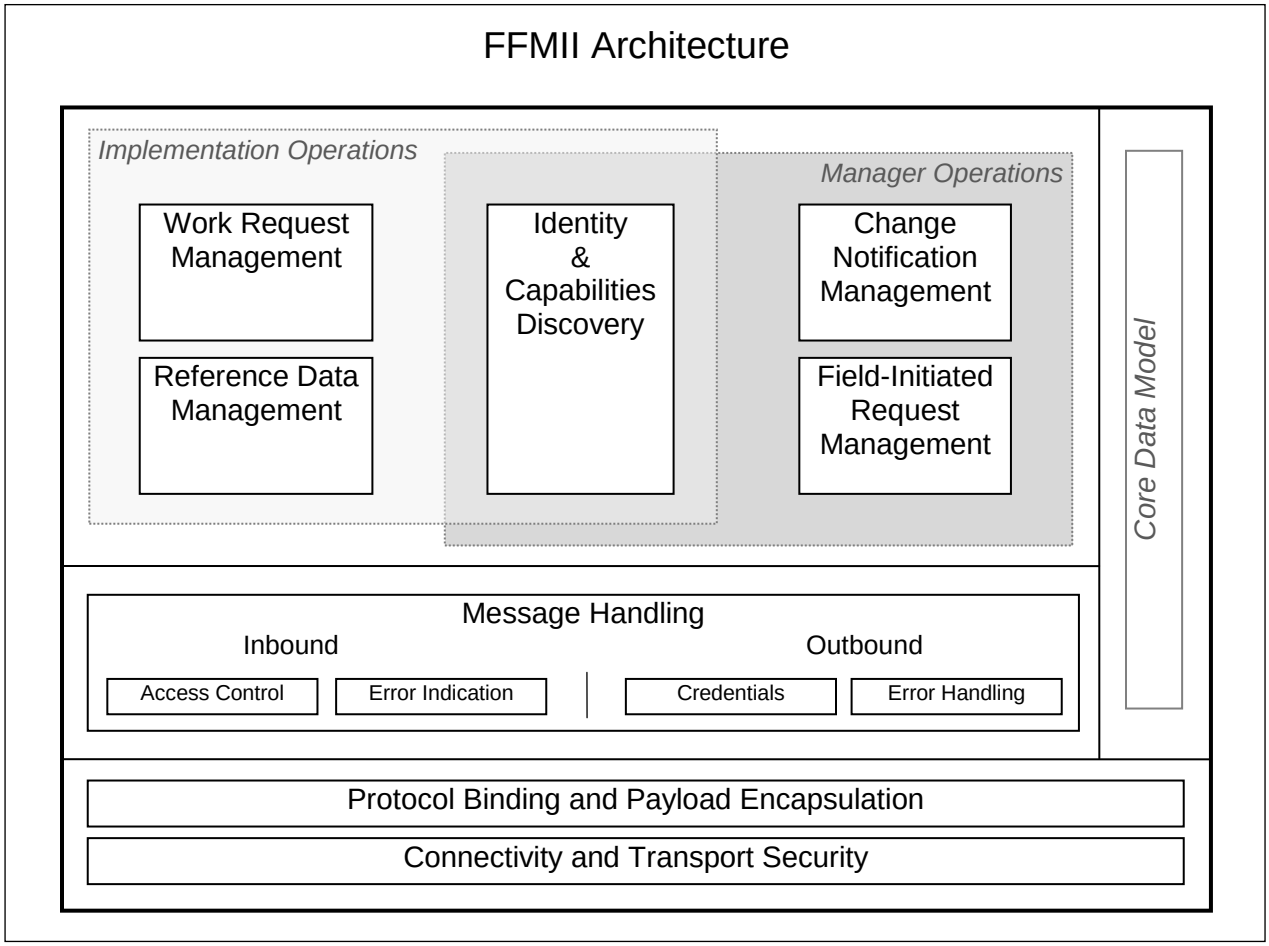


Figure 23: FFMII Architecture

As seen above, the "*Connectivity and Transport Security*" (Section 6.4.1) and "*Protocol Binding and Payload Encapsulation*" (Section 6.4.2) layers, and "*Message Handling*" (Section 6.4.3) subsystem, form a common infrastructure for Implementation and Manager to leverage when implementing functionality within their respective scope. Additionally, adherence to a common Core Data Model (Section 6.3, 7) is required on messaging and operations layers in order to ensure Implementation and Manager's interoperability across various choices of the transport technology used.

Within the operations domains layer, each domain further defines the specific type of data it processes, and functions it offers, as outlined in Figure 24:

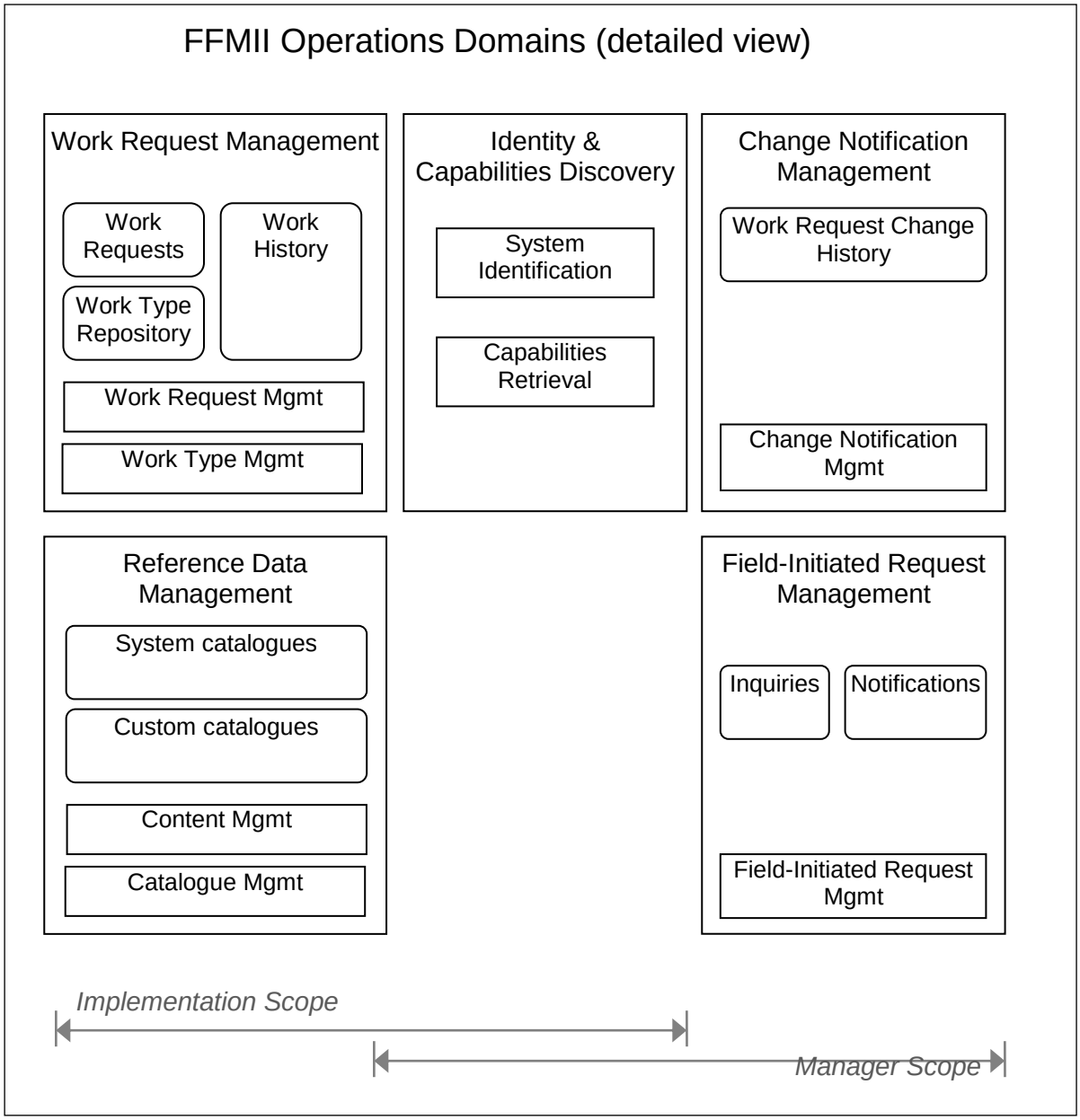


Figure 24: FFMII Operation Domains

The purpose and principal content of each operation domain is discussed in more detail throughout Section 6.2. The FFMII interface specifies a rich set of functionalities addressing a broad set of use cases. However, mandatory functionality is limited to support for Identity and Capabilities Discovery and mandatory functions of Work Request Management on Implementation side.

6.2 Operations Domains

6.2.1 Identity and Capabilities Discovery (Manager and Implementation)

The *Identity and Capabilities Discovery* area provides functionality for Implementation and Manager to expose basic descriptive information about itself. Such information includes identification of the product

and vendor, version of the supported FFMII interface variants as well as identification and profiles of provided mandatory and optional features.

An Implementation MUST always expose the Identity and Capabilities Discovery functionality to the Managers accessing it, or towards which it submits Work Request Status Change Notifications or Field-Initiated Requests. A Manager MUST expose the Identity and Capabilities Discovery functionality towards the Implementation, from where it receives Work Request Status Change Notifications or Field-Initiated Requests.

Implementations and Managers MAY expose their Identity and Capabilities Discovery functionality to any other known Managers and Implementations, respectively, given that access is properly authenticated as described in Section 6.4.3.

System Identity Descriptor:

A system Identity Descriptor provides basic descriptive data about the system. This information is primarily intended for system administrator as an additional means of ensuring authenticity of the remote end-point being accessed. Additionally, it allows an accessing a party to make certain assumptions about behavior and capabilities of the remote end beyond the scope of FFMII interface specification.

Capability Descriptors:

Capability Descriptors allow Managers and Implementations to advertise their capabilities. Capability Descriptor is structured information of pre-defined identifier informing of existence of particular capability and eventually its dimensioning and other parameters. While the mechanism for retrieving Capability Descriptors is the same for Manager and Implementation, the set of applicable capabilities differs.

Functionality of the Identity and Capabilities Discovery area is further discussed in Section 8.1. List of defined capabilities is provided in Section 8.1.

6.2.2 Work Request Management (Implementation)

Work Request Management (WRM) manages Work Requests and their updates submitted by Manager to Implementation. Each Work Request has unique identity. The Implementation maintains a Work Request Status Record that reflects the lifecycle of the Work Request as described in Section 5.1.6. Work Request Management is a mandatory capability of Implementation.

An Implementation MAY support *Work Type Repository* as an optional capability. If supported, a Manager MAY store Work Type Specifications in the Work Type Repository, and refer to those from within of Work Requests, rather than embedding a Work Type Specification in each Work Request separately.

Work Request Management is described in detail in Section 8.2.

6.2.3 Reference Data Management (Implementation)

Reference Data Management (RDM) provides a content-neutral method for Managers to deploy arbitrary data sets into the Implementation, managing content of those remotely and creating linkage (including creation of hierarchies) between individual Reference Data Items. Data managed through RDM subsystem are collectively denoted as “Reference Data”.

The primary use case of Reference Data is the ability to refer to individual RDM items or sets of items from within Work Requests, using a normalized abstract notation. For example, Reference Data can be used to specify the set of Work Types, valid values for an input Data Element or to store documents that are referred to from within Work Requests. Additionally, an Implementation may utilize Reference Data Management subsystem for exposing certain system information, such as user profile registry.

Note that the RDM subsystem of FFMII interface only defines and abstract interface for exchange of Reference Data between Managers and Implementation. It is the responsibility of Managers to keep the data up to date, while Implementation is in charge of Reference Data persistency. Any eventual distribution of Reference Data inside of Implementation (for example replication to mobile terminals) is transparent to the Manager.

Implementation MAY support Reference Data Management.

Reference Data Management is described in detail in Section 8.3.

6.2.4 Change Notification Management (Manager)

Change Notification Management allows a Manager to receive Work Request Status Change Notifications related to the lifecycle of Work Requests. Notifications are dispatched by Implementation whenever Work Request State or content change of relevance is discovered. A Manager acknowledges reception of Work Request Status Change Notifications, and processes those internally.

Manager MAY support Change Notification Management.

Change Notification Management is in more detail described in Section 8.1.3.2.1.

Alternatively, the Manager MAY poll the Implementation for changes on pending Work Requests using Work Request Management services.

6.2.5 Field-Initiated Request Management (Manager)

Field-Initiated Request Management processes Field-Initiated Requests as introduced in Section 4.2.2. Field-Initiated Requests are initiated by Assignees and are therefore asynchronous from the Manager point of view.

Field-Initiated Requests can either be *Topical Notifications*, for which no response is expected, or *Topical Inquiries,* for which a valid response or an error indication is expected.

Note that while Field-Initiated Requests are received by a Manager, the Manager MAY also act as a proxy forwarding those to other systems it is integrated with transparently from the Implementation point of view.

Manager MAY support Field-Initiated Request.

Field-Initiated Request Management is in more detail discussed in Section 8.4.

6.2.6 Response Notification Management (Implementation)

Response Notification Management allows an Implementation to receive notifications about responses to Topical Inquiries (specific type of Field-Initiated Requests) it has sent to a Manager. Field-Initiated Request Response notifications are dispatched by the Manager whenever new responses to pending Topical Inquiries become available. The Implementation acknowledges reception of notifications.

The Implementation MAY support Response Notification Management.

Alternatively, the Implementation MAY poll the Manager for new responses on pending Topical Inquiries using Field-Initiated Request Management services

6.3 Core Data Model

Core Data Model contains a common set of data types used either directly or as a base for deriving functional area specific data types throughout the rest of FFMII interface specification.

Content of Core Data Model is divided into four groups of data types:

- *primitive data types* (Section 7.2) establish a foundation out of which all other data types are built,
- *derived data types* (Section 7.3) are constrained versions of primitive data types for specific purposes and are associated with additional semantics,
- *composite data types* (Section 7.4) define more complex structures such as dictionaries and sequences, and

- *specialized data types* (Section 7.5) address specific common use cases throughout different parts of the FFMII interface specification

Each data type is described from the perspective of ifs nature and constraints, yet independently from any protocol binding technology in particular.

Core Data Model of the FFMII interface in more detail discussed in Section 7.

6.4 Common Functionality Layers

6.4.1 Connectivity and Transport Security

The *Connectivity and Transport Security* layer realizes the communication end-points of the Interface implementation. This layer provides reliable and secure means of transferring data between a Manager and an Implementation.

6.4.2 Protocol Binding and Payload Encapsulation

The *Protocol Binding and Payload Encapsulation* layer translates requests, responses, associated input parameters, and result data between the logical FFMII model and data actually transferred over the Connectivity and Transport Security layer.

FFMII SOAP Protocol Binding is described in detail in Section 9.

6.4.3 Message Handling

The *Message Handling* layer provides a unified mechanism for the Manager to invoke operations of Implementation, and vice-versa. The layer establishes common set of rules and behaviors both Manager and Implementation MUST follow when interacting with each other.

The degree of required conformance depends on the range of FFMII features Manager and Implementation realize as follows:

- Implementation MUST provide functionality of Message Handling layer for in-bound messages, and Manager MUST implement functionality of Message Handling layer for out-bound messages, since submitting Work Requests from Manager to Implementation and discovering of Implementation capabilities are fundamental mandatory use cases for FFMII.
- Manager MUST implement functionality of Message Handling layer for in-bound messages if it implements any of Identity and Capabilities Discovery (Section 6.2.1), Change Notification Management (Section 6.2.4) or Field-Initiated Request Management (Section 6.2.5) functionalities.
- Implementation MUST implement functionality of Message Handling layer for out-bound messages if it supports invocation of Identity and Capabilities Discovery, Change Notification Management or Field-Initiated Request Management functionality on Manager side.

6.4.3.1 In-bound traffic

Access Control:

Implementation or Manager accepting in-bound traffic MUST support at least one of the defined authentication methods defined in Section 9.5. To maintain an adequate level of security, an receiving party SHOULD reject authentication using system-identifier/password pairs if non-secure protocol (e.g. HTTP) is used on the transport level and network based access control is not in use.

Error Indication:

The Message Handling layer defines a common way of encapsulating error codes and error messages in operation responses. Additionally it defines a set of error codes related to the availability of the target operation to be invoked, see Section 8.1.

With every incoming request, the receiving party MUST perform access control task. Should the access be rejected, it MUST respond with corresponding E2010 error code and message.

Following a successful access control check on in-bound message, Message Handling layer MUST verify availability of the target operation being invoked. Should the operation in question be unavailable, an error message containing appropriate error code MUST be returned to the sender.

Should the invocation of the target operation finish with an error result, Message Handling layer MUST respond with corresponding error code and message.

If access control check is successful, target operation identified and available for the calling party and the operation succeeds, Message Handling layer MUST respond with “OK” response along with enclosing any data returned by the target operation.

6.4.3.2 Out-bound traffic

Credentials:

Every outbound message MUST contain valid authentication credentials, whether in the form of system-identifier/password token pair or SSL certificate. The credentials are consequently processed in receiving end by the Message Handling subsystem as described in Section 6.4.3.1.

Error Handling:

Should message transfer fail due to transient errors (network outage or remote service temporarily unavailable), the message SHOULD be scheduled for re-transmission at later moment according to defined policies. Additionally, should multiple messages be targeting same Work Request or relate to same instance of Field-Initiated Request, all such messages must be scheduled for retransmission in their original order of appearance.

Should message transfer fail due to permanent error (authentication error, invalid or unavailable target operation, or invalid operation parameters), the message MUST NOT be retransmitted anymore, as consequent attempts would result into the same errors.

7 Core Data Model

7.1 Introduction

This section describes a set of data types that are used as the foundation for all other data types throughout the FFMII interface specification. The minimum requirements and the general characteristics of each data type are described throughout this, while the way the corresponding values are expressed for the purpose of message exchange depends in the protocol binding in use.

FFMII Core Data Model divides into primitive data types, derived data types, composite data types and specialized data types as described in the following sections.

7.2 Primitive Data Types

Primitive data types served as foundation for building specialized derived and composite data types used throughout the FFMII interface specification. Table 2 specifies the primitive data types used in this specification.

Type	Description	Notes
String	Sequence of zero or more UNICODE characters	As defined by [Schema2]
Integer	Signed integer value	As defined by [Schema2]
Double	Double-precision floating point value	As defined by [Schema2]
Decimal	Arbitrary-precision decimal value	For example, monetary values; As defined by [Schema2]
Boolean	True or False	As defined by [Schema2]
Date	Date information comprising of year, month and day of month	As defined by [Schema2]; See note below regarding time zones.
Time	Time of day comprising of hour, minute and second information, and optionally including time zone offset relative to UTC [Schema2]	
DateTime	Combined date and time value following requirements for “Date” and “Time” data types	
Duration	Represents time duration	
Binary	Sequence of zero or more binary digits	Opaque binary data, corresponding to binary data types defined in [Schema2]

Table 2: Primitive Data Types

Notes:

1. Time zones:
    a. When generating messages, the sender MAY specify Date, Time, and DateTime values with or without time zone information (rationale: since a common use case is for all work

to be in the same time zone; presumably Implementations will have an internal configuration option for setting the default time zone).

b. When receiving messages, the receiver MUST correctly process Date, Time, and DateTime values both when given with time zone information and when given without it (in the latter case, correct processing means using a default time zone that is implementation-defined).

2. While definitions use the XML Schema, they may be used in any protocol binding.

7.3 Derived Data Types

Derived data types are based on primitive data types with additional constraints and semantics applied to them. The Table 3 describes specialized core data types of FFMII interface:

Type	Base Type	Description
Identifier	String	Sequence of characters used as identifier of data objects or for reference purposes An Identifier consists of one or more alpha-numeric characters from [ISO/IEC 8859-1], underscore (_) or dot (.), where first character is an underscore or alpha character, and last character is not dot.
ErrorCode	String	Upper case letter 'E' followed by four decimal digits. For example, “E0000”, “E1234”, etc.
LocaleSpecifier	String	A locale specifier consisting of a lower-case two-letter language code as defined by [ISO-639] optionally followed by an underscore (_) and an upper-case two-letter country code as defined by [ISO-3166] Example: “en”, “en_US”, “fi_FI”.

Table 3: Derived Data Types

7.4 Composite Data Types

Composite data types are structures composed of primitive data types and multiplicity rules. The Table 4 describes composite core data types of FFMII interface:

Type	Description	Notes
Class	Fixed set of named properties of specified type (primitive, derived, specialized or composite)	
Dictionary	Dynamic collection of named primitive (key/value pairs), order of which is not significant	
Sequence	Dynamic collection of unnamed items of a specified type (primitive, derived, specialized or composite), order of which is significant unless otherwise specified	

Table 4: Composite Data Types

7.5 Specialized Data Types

7.5.1 Multi-Language Text (MLText)

MLText is a sequence of data elements containing one or more translations of the same textual information.

MLText			
Property	**Type**	**M/O**	**Description**
Values	Sequence of LocalizedString	M	Alternative localized strings

LocalizedString			
Property	**Type**	**M/O**	**Description**
Locale	LocaleSpecifier	O	Locale this value is associated with. If not specified, the associated value is used as the default value unless more specific match is found.
Value	String	M	The localized string value

Table 5: Specialized Data Types

The receiver of MLText data SHOULD display to user the most specific translation that matches the user preferences or otherwise the default alternative. Default alternative is the one for which no locale is specified, or if no such alternative exists, the first defined alternative.

7.5.2 Location

Location is a data element representing a geographical location.

Location			
Property	**Type**	**M/O**	**Description**
Longitude	Decimal	M	Longitude of location
Latitude	Decimal	M	Latitude of location
Altitude	Decimal	O	Altitude of location

Table 6: Location Data Type

The coordinate system used for the location values is defined in [KML].

7.5.3 MultiChoiceAlternative

MultiChoiceAlternative declares a valid value for a Data Field and MAY specify an associated label to be used for display purposes.

MultiChoiceAlternative			
Property	**Type**	**M/O**	**Description**

MultiChoiceAlternative			
Property	**Type**	**M/O**	**Description**
Label	MLText	O	Label to be displayed for this alternative in the user interface. If not specified then the value is used as the label. Label may include data variables
Value	DataValue	M	Valid value for this element.

Table 7: MultiChoiceAlternative Data Type

7.5.4 Custom Properties

Some data types (classes) defined through FFMII interface specification MAY contain a specific property called “CustomProperties” of type Dictionary. CustomProperties, if allowed within particular scope, are intended for exchange of implementation-specific information. As per the definition of the Dictionary type, custom properties are key-value pairs, and only values of Primitive or Derived Data Types may be used.

Integrated software components are not necessarily aware of all implementation-specific features supported by the counter-parts. The following interoperability rules MUST be followed to ensure consistent behavior across Managers and Implementations of different origins:

- Manager or Implementation MUST NOT supply or accept “CustomProperties” in association with data types for which no such property is defined
- Manager and Implementation MUST implement the following interoperability rules and logic for any data type for which “CustomProperties” property is defined
    - if unsupported but grammatically valid custom properties are received, the receiving end MUST successfully accept the content as opaque data ignoring any semantics the content may have for the supplying end. The receiving end MAY warn about the unsupported content in the operation response.
    - if supported custom properties are received, the receiving end MUST validate the content based on the known semantics and signal an error if the supplied content is invalid.
    - any accepted (supported or unsupported) custom properties supplied as part of an object that is exposed for retrieval through the FFMII interface MUST be stored with the object itself, and returned whenever the object is retrieved in consequent operations.

8 Data Types and Operations

8.1 Identity and Capabilities Discovery

8.1.1 Introduction

Identity and Capabilities Discovery subsystem allows the remote party to verify identity of the FFMII end-point it is accessing, and retrieve capabilities and operational constraints of the remote end.

Implementation MUST support Identity and Capabilities Discovery. Manager MUST support Identity and Capabilities Discovery if Manager provides services to Implementation.

8.1.2 Data Types

8.1.2.1 Identity Descriptor

System Identity Descriptor provides basic information about the end-point being accessed:

IdentityDescriptor			
Property	**Type**	**M/O**	**Description**
SystemType	Identifier	M	“ERMS” if the target system is Manager, or “FFMS” if the target system is Implementation
SystemId	Identifier	O	Identifier of the system being accessed
ProductId	Identifier	O	Unique identifier of the product constructed according to convention specified later in this section
Properties	Dictionary	M	Standard attributes providing additional information about the system being accessed
CustomProperties	Dictionary	O	Implementation-specific attributes providing additional information about the system being accessed outside of the scope of FFMII interface

Table 8: IdentityDescriptor

SystemId

SystemId is an identifier for distinction between several systems of the same type. Usage of the identifier is optional. The purpose of this identifier is to bring an additional level of confidence particularly during early phase of system-to-system integrations.

ProductId

ProductId is a unique identifier of the product in question. Uniqueness of the identifier is the sole responsibility of the product vendor. However, the identifier MUST begin with an element (vendor tag) that uniquely identifies the vendor world-wide (such as domain name, registered trademark or registered project name). A Vendor MAY further refine the ProductId by including elements identifying specific product or version.

Based on the discovered ProductId, the accessing party MAY make use of implementation-specific features or other assumptions beyond the scope of FFMII specification.

Note: Extensive usage of Product identifier for decision making on the caller side may result in unwanted dependencies between ERMS and FFMS, and therefore must be considered carefully in advance

Standard properties

Properties element of System Identity Descriptor contains standard attributes that further describe the end-point.

Table 9 specifies System Identity Descriptor standard properties recognized by this version of FFMII interface specification:

Property name	Type	M/O	Description
ProductName	String	O	Name of the product whose services are being accessed
ProductVersion	String	O	Version of the product whose services are being accessed
VendorName	String	O	Name of the vendor of the product

Table 9: Standard Properties

8.1.2.2 Capability Descriptor

Capability Descriptor conveys details of a specific capability, identified through a standard identifier. The capability is typically optional, provided by an Implementation or a Manager. In specified cases, Capability Descriptors MAY expose additional details of mandatory capabilities, e.g. override or otherwise refine default parameters.

If a Manager or an Implementation provides a Capability, it MUST expose the corresponding Capability Descriptor. If a Manager or an Implementation does not provide a Capability, it MUST NOT expose the corresponding Capability Descriptor. In other words, the existence of a Capability Descriptor signifies support for the corresponding Capability. Table 10 specifies elements of Capability Descriptor:

CapabilityDescriptor			
Property	**Type**	**M/O**	**Description**
Id	Identifier	M	Unique identifier of the capability. For standard capabilities see Table 11.
Description	String	O	Textual description of the capability for informational purposes
Properties	Dictionary	O	Additional information regarding the capability in question (for example, sizing limits)
CustomProperties	Dictionary	O	Implementation-specific attributes that provide additional information about the capability

Table 10: Capability Descriptor

Some Capability Descriptors have no mandatory properties, which are signified by '--' in Property, Type, M/O and Description columns of the Capability Descriptor. For example, see Section 8.1.3.1.1. Any Capability Descriptor MAY specify CustomProperties to convey implementation specific attributes.

8.1.3 Standard capabilities

Table 11 describes the standard capabilities for Implementation and Manager.

Capability ID	Description	Provider	M/O
WRM	Work Request management	Implementation	M

Capability ID	Description	Provider	M/O
RDM	Reference Data Management	Implementation	O
RDM.Users	User Information Repository	Implementation	O
RDM.WorkTypes	Work Type Repository	Implementation	O
RDM.FIR	Field-Initiated Requests Repository	Implementation	O
RDM.Custom	Custom Reference Data Repositories	Implementation	O
Client.Webui.Handset	Web UI enabled mobile client	Implementation	O
Client.Webui.Desktop	Web UI enabled desktop client	Implementation	O
Client.Native	Native client realization	Implementation	O
System.Locales	Supported locales	Implementation	O
CNM	Change Notification Management	Manager	O
FIRM	Field-Initiated Request Management	Manager	O

Table 11: Standard Capabilities

8.1.3.1 Implementation capabilities

8.1.3.1.1 WRM

An Implementation MUST support WRM capability.

Capability Descriptor				
Identifier	WRM			
Description	Work Request Management			
Property	**Type**	**M/O**	**Default**	**Notes**
--	--	--	--	--

8.1.3.1.2 RDM

This Capability Descriptor informs of Reference Data Management (see Section 8.3) Capability provided by the Implementation.

Capability Descriptor				
Identifier	RDM			
Description	Reference Data Management			
Property	**Type**	**M/O**	**Default**	**Notes**
--	--	--	--	--

8.1.3.1.3 RDM.Users

User repository stores information about Assignees known to Implementation, such as user names and credentials, user authorization for system and user administration, locale settings, and other types of preferences. The user identifier uniquely identifies the user for the purpose of assigning Work Requests

through Work Request Management operations. When exposed as part of RDM, the content of the user profiles repository can be accessed and/or managed through RDM operations. In addition to rights specified in the Repository Descriptor, the further restrictions can be specified, see Capability Descriptor below. If RestrictedRead and/or RestrictedWrite properties are False or not specified, no access constraints between Managers are imposed on corresponding operations.

Capability Descriptor				
Identifier	RDM.Users			
Description	User Profile Repository			
Property	**Type**	**M/O**	**Default**	**Notes**
RestrictedRead	Boolean	O	False	If True, Manager can only read entries it has created (while it still can refer to other entries by their identifiers)
RestrictedWrite	Boolean	O	False	If True, Manager can only update or remove entries it has created

Note: Structure of Users entries is discussed in Section 8.9.

8.1.3.1.4 RDM.WorkTypes

This capability signals that the Implementation supports the Work Type Repository, as described in Section 8.10

Work Type Repository contains structural definitions of pre-defined Work Types. Work Type definitions do not include Work Request instance specific data. Work Requests can refer to Work Type entries. This eliminates the need to include full Work Request structure with each Work Request.

Capability Descriptor				
Identifier	RDM.WorkTypes			
Description	Work Type Repository			
Property	**Type**	**M/O**	**Default**	**Notes**
WriteProtected	Boolean	O	False	If True, Manager cannot update content of the repository in any way

8.1.3.1.5 RDM.FIR

FIR repository stores information about Field-Initiated Requests made available by the Manager to the Implementation.

Capability Descriptor				
Identifier	RDM.FIR			
Description	Field-Initiated Requests Catalogue			
Property	**Type**	**M/O**	**Default**	**Notes**
--	--	--	--	--

1139 Note: Structure of FIR entries is discussed in Section 8.118.11.

1140 8.1.3.1.6 RDM.Custom

1141 An Implementation uses this Capability Descriptor to specify constraints of management of Custom
1142 Reference Data Repositories through RDM.

Capability Descriptor				
Identifier	RDM.Custom			
Description	Custom Reference Data Repositories			
Property	**Type**	**M/O**	**Default**	**Notes**
WriteProtected	Boolean	O	False	If True, a Manager can obtain a list of all Custom Reference Data Repositories of the Implementation, but it can neither remove nor create new Custom Reference Data Repositories

1143

1144 8.1.3.1.7 Client.Webui.Handset

1145 Through this capability, an Implementation indicates that it supports web browser-enabled mobile devices
1146 as one of the client realization technologies. From the FFMII standpoint, this capability has indicative
1147 value only, since the Interface specification is neutral with respect to the type of clients and their
1148 implementation details. Nevertheless, an Implementation may use vendor specific attributes
1149 (CustomProperties) for disclosing implementation details of the client.

Capability Descriptor				
Identifier	Client.Webui.Handset			
Description	Web UI enabled mobile client			
Property	**Type**	**M/O**	**Default**	**Notes**
--	--	--	--	--

1150 8.1.3.1.8 Client.Webui.Desktop

1151 Through this capability, an Implementation indicates that it supports web browser-enabled desktop
1152 computers and tablets as one of the client realization technologies. From the FFMII standpoint, this
1153 capability has indicative value only, since the Interface specification is neutral with respect to the type of
1154 clients and their implementation details. Nevertheless, an Implementation MAY use vendor specific
1155 attributes (CustomProperties) for disclosing implementation details of the client.

Capability Descriptor				
Identifier	Client.Webui.Desktop			
Description	Web UI enabled desktop client			
Property	**Type**	**M/O**	**Default**	**Notes**
--	--	--	--	--

8.1.3.1.9 Client.Native

Through this capability, Implementation indicates that it supports native-code mobile client as one of the client realization technologies. From the FFMII standpoint, this capability has indicative value only, since the Interface specification is neutral with respect to the type of clients and their implementation details. Nevertheless, an Implementation MAY use vendor specific attributes (CustomProperties) for disclosing implementation details of the client.

Capability Descriptor				
Identifier	Client.Native			
Description	Native client implementation			
Property	**Type**	**M/O**	**Default**	**Notes**
--	--	--	--	--

8.1.3.1.10 System.Locales

Through this capability an Implementation specifies the set of locales it supports and is able to relay to the end user. ERMS MAY use this information, for example, for reducing the amount of data included in Work Requests when the Implementation supports only a subset of locales ERMS is able to generate data for.

NOTE: an *Implementation MUST be able to receive data for any locale, even if it is not necessarily able to display that data to user properly.*

If this capability is not specified, the extent of the locale support is not defined and not known to ERMS. If the locales are not specified, ERMS MUST NOT assume that a specific locale is supported but it MAY send localized data targeted for any locale.

Capability Descriptor				
Identifier	System.Locales			
Description	Supported locales			
Property	**Type**	**M/O**	**Default**	**Notes**
supported-locales	String	M	--	String containing a list of supported Locale Identifiers (See Section 7) separated with a single space. If only the language part is present then any country variant is supported for the language. Example: “en_US en_UK” Implementations supporting this capability MUST advertise at least one supported locale.

8.1.3.2 Manager capabilities

8.1.3.2.1 CNM

This Capability Descriptor informs of Change Notification Management (see Section 6.2.4) functionality provided by the Manager

Capability Descriptor				
Identifier	CNM			
Description	Change Notification Management			
Property	**Type**	**M/O**	**Default**	**Notes**
largest-attachment	Integer	O	--	Maximal size of an attachment the Manager is able to accept as part of Change Notification message. The value is provided in kilobytes (1kB = 1024 B). If the property is not specified, the maximal size of an attachment is not known or is not constrained.
allow-wr-bundling	Boolean	O	True	If True, a single notification sent by Implementation MAY contain changes related to several different Work Requests

8.1.3.2.2 FIRM

This Capability Descriptor informs of Field-Initiated Request Management (see Section 6.2.5) functionality provided by the Manager. .

Capability Descriptor				
Identifier	FIRM			
Description	Field-Initiated Request Management			
Property	**Type**	**M/O**	**Default**	**Notes**
--	--	--	--	--

8.1.4 Operations

Identity and Capabilities Discovery subsystem exposes the following operations for usage by remote clients:

Operation	Description	Notes

SYS_INFO_GET	Returns System Identity Descriptor for the end-point being accessed
	
	`Input parameters:`
	`None`
	
	`Return value:`
	`IdentityDescriptor`	

SYS_CAPA_GET	Returns sequence of System Capability Descriptors exposed by the end-point being accessed	

Operation	Description	Notes
	Input parameters: None Return value: Sequence of CapabilityDescriptor	

8.2 Work Request Management

8.2.1 Data Types

This section specifies the data types associated with Work Request Management. The main data types are Work Type, Work Request and Work Request Status Record. They are specified in dedicated sections together with additional data types they refer to.

8.2.1.1 Work Type

A Work Type specifies the structure of a particular type of Task. It specifies the flow of work, possible States, data that needs to be provided with a Work Request, data to be collected from the Assignee during the work flow, and static data common to all Work Requests of the particular type.

A Work Type is described by a Work Type Specification that is either stored in the Work Type Repository or provided in-line with a Work Request. Several Work Requests may share the same Work Type Specification by referring to it.

WorkType (abstract base class)			
Property	**Type**	**M/O**	**Description**
			No common properties

Table 12: WorkType

WorkTypeReference (extends WorkType)			
Property	**Type**	**M/O**	**Description**
Id	Identifier	M	Identifier of the Work Type stored in the Work Type Repository

Table 13: WorkTypeReference

WorkTypeSpecification (extends WorkType)			
Property	**Type**	**M/O**	**Description**
SharedDataElements	Sequence of DataElementSpecification	O	Shared Data Element Specifications that may be referred to from within Data Forms specified in this Work Type Specification. See Section 8.5
SharedActions	Sequence of ActionSpecification	O	Shared Actions that may be referred to from within Actions property of a Step Specification or invoked by the Manager via the WR_ACTION operation. See Section 8.2.1.3 for details
SharedStateModels	Sequence of StateModelSpecification	O	Shared state model specifications that may be referred to from within ActivitySpecifications of this Work Type Specification. See Section 8.2.1.2 for details
Header	DataForm	M	Contains Work Request summary information intended to be used in places, such as Work Request list, where possibly several Work Requests are presented to the user All elements in this form must be declared as for displaying only (i.e. not updateable)
Overview	DataForm	M	Specifies the Data Elements that describe the work on general level. These are used when presenting an overview of the work to the Assignee. See Section 8.5 for details
Instructions	DataForm	O	More detailed work instructions that should be combined with possible Step-specific instructions. See Section 8.5 for details
Activities	Sequence of ActivitySpecification	M	One or more Activities that are part of this type of work
CustomProperties	Dictionary	O	Implementation-specific custom properties

1200 **Table 14: WorkTypeSpecification**

1201 ### 8.2.1.2 Activity, State and Step

1202 The execution model for a Task is described in Section 5.1.2.4. This section specifies the related data
1203 types Activity, State and Step.

1204 An Activity is a distinct part of work associated with a Task. Multiple Activities may be used to model parts
1205 of work that are, for example, performed in different locations, are distinct phases that are not necessarily
1206 executed sequentially, or are performed by different Assignees.

ActivitySpecification

Property	Type	M/O	Description
Id	Identifier	M	Unique identifier in the context of Activities specified in the Work Type Specification
StateModel	StateModel	M	Specifies the Activity State Model for this Activity. See section 5.1.2.4.
EnableCondition	Expression	O	Specifies the Boolean Expression that represents the Enable Condition of this Activity See Section 5.1.2.5 on modeling dependencies and Section 8.6 for expression structure. If EnableCondition is not specified, the Activity may proceed independently at any time.
ActivityLocationData	DataForm	O	Data elements providing information the Assignee will need to reach the location associated with the Activity. See Section 5.1.2.3.
Supplementary	Boolean	O	Indicates whether the Activity is Supplementary. If this is not specified, the Activity is considered to be non-Supplementary.
CustomProperties	Dictionary	O	Implementation-specific custom properties

1207 **Table 15: Activity Specification**

1208 The *State Model* of an Activity may be specified either as an in-lined *State Model Specification*
1209 dynamically specifying the custom States, Steps and Actions or by referring to a State Model specified in
1210 the SharedStateModels property of the Work Type Specification,

StateModel (abstract base class)			
Property	**Type**	**M/O**	**Description**
			No common properties

1211 **Table 16: StateModel**

1212

StateModelReference (extends StateModel)			
Property	**Type**	**M/O**	**Description**
Id	Identifier	M	Identifier of a State Model specified in the SharedStateModels property of the Work Type Specification.

1213 **Table 17: StateModelReference**

1214

StateModelSpecification (extends StateModel)			
Property	**Type**	**M/O**	**Description**
Id	Identifier	M	Unique identifier in the context of SharedStateModels of the Work Type Specification, or any identifier if in-lined with an Activity Specification.
Steps	Sequence of StepSpecification	M	Describes all the Steps the associated Activity has. Work flow becomes defined through Actions associated with Steps. The order in which the Steps have been listed in this property is not significant.
States	Sequence of StateSpecification	M	Describes all the possible States the associated Activity can be in. The order in which the States have been listed in this property is not significant.
InitialStepId	Identifier	M	Identifier of the initial Step, referring to one of the Steps defined within the States property of this model

Table 18: StateModelSpecification

A State Specification describes a possible top level State a particular Activity can be in. Possible States can be specified separately for each Activity type and they should describe the States of work the associated Manager is interested in. See Section 5.1.2.4 for an example of a State Model. The background idea is that each State is further divided to one or more Steps that describe the micro flow of work associated with the State and provide working instructions to the Assignee. Formally each Step is associated with a State. Thus, Steps associated with the same State form the State. Steps specify the possible transitions and provide work Instructions for the Step.

StateSpecification			
Property	**Type**	**M/O**	**Description**
Id	Identifier	M	Unique identifier in the context of States specified in the Activity Specification
Label	MLText	M	Label for this State for user-interface purposes. Label may include data variables (see Section 8.5.10 Data Variables)
StatusCategory	Identifier	M	The Status category that this State belongs to, “Open”, “Active”, “Inactive”, or “Closed”. See Section 5.1.2.4
StatusIndicator	Identifier	O	The Status Indicator associated with this State, if any. See Section 5.1.2.4

Table 19: StateSpecification

Step specification describes one Step to be performed by an Assignee.

StepSpecification			
Property	**Type**	**M/O**	**Description**
Id	Identifier	M	Unique identifier in the context of Steps specified in the Activity Specification
StateId	Identifier	M	Identifier of associated State.
Title	MLText	O	Short title describing what the user is expected to do in this Step. This should either be in an imperative form or describe the overall State of the Activity, such as “Completed”. Title may include data variables (see Section 8.5.10).
Instructions	DataForm	O	Specifies the Step Instruction for this Step. See section 5.1.2.3.
Actions	Sequence of Action	O	Defines the Actions available to the Assignee in this Step. The first Action is the default for user-interface purposes
AvailableTopics	Sequence of AvailableTopic	O	Defines the FIR Topics available to the Assignee

1227 **Table 20: StepSpecification**

1228 ### 8.2.1.3 Action

1229 The execution model for a Task is described in Section 5.1.2.4. This section describes the data types
1230 used to specify *Actions*.

1231 Actions may be specified in-line in Step Specifications using *Action Specification* or they can be specified
1232 in SharedActions property of the Work Type Specification and referred to using *Action Reference.*

Action (abstract base class)			
Property	**Type**	**M/O**	**Description**
			No common properties

1233 **Table 21: Action**

ActionReference (extends Action)			
Property	**Type**	**M/O**	**Description**
Id	Identifier	M	Identifier of an Action specified in SharedActions of the Work Type Specification

1234 **Table 22: ActionReference**

1235

ActionSpecification (extends Action)			
Property	**Type**	**M/O**	**Description**
Id	Identifier	M	Unique identifier in the context of the associated Work Type Specification
Label	MLText	M	Label for this Action for user-interface purposes. Label may include data variables (see Section 8.5.10 Data Variables)
Keyword	MLText	O	Short keyword or mnemonic to be used in space-limited used interfaces, for example SMS
GenericType	Identifier	O	Pre-defined generic Action type. Can be used as hint for the rendering of the user interface view (for example inclusion of graphical Action buttons instead of textual descriptions). If present, GenericType MUST be one of "`Accept`", "`Reject`", "`Start`", "`Suspend`", "`Resume`", or "`Complete`" corresponding to an Action that accepts, rejects, starts, suspends, resumes, or completes Activity, respectively
ConfirmRequired	Boolean	O	Specifies if the user interface SHOULD confirm whether the user really wants to execute this Action. If False, confirmation of the Action is not required. If True, the user interface SHOULD confirm performing the Action. Default is False, Typically ConfirmRequired is set True in association with irrevocable Actions that should be confirmed.
EnableCondition	Expression	O	Specifies the Boolean Expression that represents the Enable Condition of this Action. See Section 5.1.2.5 on modeling dependencies and Section 8.6 for expression structure. If an EnableCondition is not specified, the Action is Enabled.
AvailableToAssignee	Boolean	O	Specifies whether this Action can be invoked by the Assignee. If False, then this Action can only be invoked by the Manager via the FFMII Interface. This can be used to specify special State transitions that are not available to the Assignee. Default is True.

InputForm	DataForm	O	Input Form that needs to be filled as part of executing this Action. The Implementation MUST validate the input form before committing on this Action. See Section 5.1.4.
NextStepId	Identifier	M	Identifier of the next Step specified within this Work Type Specification. A special reserved value of “PopStepStack” indicates transition to the Step retrieved from the top of the Step Stack (see Section 8.2.1.8)
PushStep	Boolean	O	Whether to push the identifier of the current Step to the top of the Step Stack (see Section 8.2.1.8). Default is False.
AssertAdministrativeClosingStatusId	Identifier	O	If any value is specified, then executing this Action asserts the Administrative Closing Status for the associated Work Request. The specified identifier becomes the Adminstrative Closing Status value. See Section 5.1.5 for details
DataUpdateOperations	Sequence of DataUpdateOperation	O	Specifies the data update operations to be performed on the Work Request when this Action is invoked. See Section 8.2.1.5 for details
CustomProperties	Dictionary	O	Implementation-specific custom properties

Table 23: ActionSpecification

8.2.1.4 Available Topics

Topics on which Assignee may invoke Field-Initiated Requests in a particular Step of a Work Request are defined in a Step Specification (See Section 8.2.1.2) as a sequence of *Available Topic* instances.

An Available Topic binds a specific Topic to a Step and MAY impose additional conditions on when the Topic is enabled.

AvailableTopic			
Property	**Type**	**M/O**	**Description**
TopicId	Identifier	M	Identifier of a Topic on which the Assignee may invoke Field-Initiated Requests
EnableCondition	Expression	O	Boolean Expression specifying if this Topic is enabled and available to the Assignee. See Section 8.6 for Expressions. If an EnableCondition is not specified, the Topic is Enabled.

Table 24: AvailableTopic

8.2.1.5 Data Update Operations

In addition to storing any provided user input data and supplementary input data to the Work Request Status Record, an Action may also update Work Request data directly. This feature is designed to enable instant user interface response on certain types of user input where otherwise interaction with upstream system (ERMS) would be required, and which would therefore possibly suffer from inherent round-trip delays between FFMS and ERMS.

Data elements introduced in this chapter are used to specify data update operations associated with Actions. Both user-provided input data and specified supplementary data are interpreted equally when performing the update operations.

If Work Request data is updated via such operations, the updated Data Element values are stored in the *UpdatedData* property of the associated *Work Request Status Record*, thereby made visible to the ERMS system for retrieval (and possible consolidation) at any point of time.

Data Update Operation is the common base class for different kinds of update operations.

DataUpdateOperation (abstract base class)			
Property	**Type**	**M/O**	**Description**
TargetElementId	Identifier	M	Identifier of the target element, i.e. the Work Request Data Element to be updated

Table 25: DataUpdateOperation

Set Value Operation causes the value of the target element to be set to the evaluated value of the specified Expression which MAY refer to Work Request data or Action input data.

SetValueOperation (extends DataUpdateOperation)			
Property	**Type**	**M/O**	**Description**
ValueExpression	Expression	M	Expression evaluated to obtain the value to be set to the target element. The evaluated value MUST be compatible with the target element specification.

Table 26: SetValueOperation

Add Row Operation causes a new row to be added to the specified Data Matrix using the evaluated values of specified Expression as values for the new data row.

AddRowOperation (extends DataUpdateOperation)			
Property	**Type**	**M/O**	**Description**
ValueExpressions	Sequence of Expression	M	Sequence of Expression to be evaluated to obtain values for the new data row, in the order the columns were specified. Each evaluated value MUST be compatible with the respective column specification.
InsertPosition	Identifier	O	Token indicating at which position the new item should be included Valid values are: “First” (inserted in front of first value in the data matrix, if any) “Last” (appended after the last item in the data matrix, if any) Default is “Last”.

Table 27: AddRowOperation

8.2.1.6 Work Request

Each Activity of a Work Request MAY be assigned to an Assignee. Note that different Activities MAY be assigned to different Assignees. A Work Request refers to or includes a Work Type Specification for specification of the work flow and Data Elements and provides data values to be bound to the Data Elements declared in the specification.

The Implementation MUST maintain the following data for each Work Request throughout the entire lifecycle of the request:

- Work Type Specification (see Section 8.2.1.1) associated with the request as it was at the request creation time (changes to the specification within the Work Type Repository MUST NOT affect existing Work Requests referencing it)
- Work Request Status Record (see Section 8.2.1.9)
- Step Stack (see Section 8.2.1.8)

WorkRequest			
Property	**Type**	**M/O**	**Description**
Id	Identifier	M	Unique identifier in the context of the Manager
WorkType	WorkType	M	Work Type specific template describing the work. May either be a reference to the Work Type Repository, or include an in-lined Work Type Specification specifically for this request
WorkRequestData	Sequence of DataBinding	M	Data content specified by the Work Type Specification to be supplied in-lined with each Work Request. Also values for data variables can be set here. The order in which bindings are listed in this property is not significant. See Data Form Sections 5.1.4 for details
ActivityData	Sequence of ActivityDataRecord	O	Activity-specific data. The order in which records are listed in this property is not significant. Each record identifies the associated Activity by its identifier. There MUST NOT be multiple records associated with the same Activity.
PriorityIndicator	Int	O	Relative priority of this request in range 1 to 10, 1 being the highest priority. Work for which priority is not specified is lower priority than any work for which priority is specified.
CustomProperties	Dictionary	O	Implementation-specific custom properties

1278 **Table 28: WorkRequest**

1279 Type *ActivityDataRecord* is used to specify or override Activity-specific data.

1280

ActivityDataRecord			
Property	**Type**	**M/O**	**Description**
Id	Identifier	M	Identifier of the Activity for which data is being provided.
AssigneeId	Identifier	O	Identifier of the Assignee this Activity is assigned to. Refers to the Assignees stored in the “Users” repository. If not specified then the associated Activity is not currently assigned to any Assignee.
Schedule	ScheduleSpecification	O	Defines details on when the work is to be done

1281 **Table 29: ActivityDataRecord**

8.2.1.7 Schedule Specification

The Schedule associated with an Activity has two logical parts: The time constraints defining when the Activity may be executed; and the planned time for executing the Activity. See Section 5.1.3 for definitions.

ScheduleSpecification			
Property	**Type**	**M/O**	**Description**
LatestStart	DateTime	M	The latest time when the Activity may be started
EarliestStart	DateTime	O	The earliest time when the Activity may be started. If EarliestStart is not specified, it is assumed that the Activity may be started at any time between the present and the LatestStart
LatestFinish	DateTime	O	The latest time when the Activity may be finished. If LatestFinish is not specified, the finishing time is not constrained
AppointmentStart	DateTime	O	Start time of the appointment time window during which the service provider has promised that the service would be delivered. AppointmentStart and AppointmentFinish MUST be specified together as a pair.
AppointmentFinish	DateTime	O	Finish time of the appointment time window during which the service provider has promised that the service would be delivered. AppointmentStart and AppointmentFinish MUST be specified together as a pair.
Duration	Duration	O	Estimated duration of work to be done.
PlannedStart	DateTime	O	Planned Start time of the Activity
PlannedFinish	DateTime	O	Planned Finish time of the Activity. If PlannedFinish is specified, PlannedStart MUST be specified as well

Table 30: ScheduleSpecification

8.2.1.8 Step Stack

The Implementation MUST maintain a Step Stack for each Activity. The Step Stack is a stack of Step identifiers. New identifiers that need to be preserved are always stored (“pushed”) to the top of the stack, and top-most identifiers may be removed (“popped”) from the top of the stack when needed.

Initially the Step Stack is empty but if a user takes an Action with the PushStep property set to True (see Section 8.2.1.3), the identifier of the current Step is stored on the top of the Step Stack. Another Action may later pop the top-most identifier from the stack to return to the pushed Step.

The Step Stack makes it possible, for example, to introduce a generic State and Step for temporarily suspending the work and yet being able to later return to the Step where the interruption occurred.

8.2.1.9 Work Request Status Record

Work Request Status Record contains information about the current status of the associated Work Request as well as history of state transitions and Work Request data updates.

WorkRequestStatusRecord			
Property	**Type**	**M/O**	**Description**
WorkRequestId	Identifier	M	Identifier of the associated Work Request
StatusSnapshot	StatusSnapshotRecord	M	Current status of the Task
ChangeHistory	Sequence of ChangeHistoryEntry	M	Sequence of zero or more Change History Entries in chronological order
CustomProperties	Dictionary	O	Implementation-specific custom properties

1300 **Table 31: WorkRequestStatusRecord**

1301 *Status Snapshot Record* contains information about the current status of a Work Request.

StatusSnapshotRecord			
Property	**Type**	**M/O**	**Description**
RevisionNumber	Int	M	Monotonically increasing revision number starting from zero and incremented each time the status record of this request is updated (that is, for each invoked Action and each time Work Request data content is updated)
RevisionTime	DateTime	M	When this status record was last updated (time of last update or initial creation of the request)
CurrentTaskStatusId	Identifier	M	One of the Status Categories "*Open*", "*Active*", "Inactive", or "*Closed*" reflecting overall Task Status as defined in Section 5.1.5
CurrentTaskStatusEnterTime	DateTime	M	When the current Task Status as indicated by CurrentTaskStatusId was entered
AdministrativeClosingStatusId	Identifier	O	The asserted Administrative Closing Status value, if any. See Section 5.1.5.
ActivityStatusInfo	Sequence of ActivityStatusRecord	M	Status information for all associated Activities. The order in which the records are listed in this property is not significant. Each record identifies the associated Activity by its identifier. There MUST be exactly one status record for each Activity.

1302 **Table 32: StatusSnapshotRecord**

1303 *Activity Status Record* contains information about the current status of a specific Activity.

ActivityStatusRecord			
Property	**Type**	**M/O**	**Description**
ActivityId	Identifier	M	Identifier of the associated Activity

CurrentActivityStateId	Identifier	M	Identifier of the current State of the Activity State Model
CurrentActivityStateEnterTime	DateTime	M	Timestamp of when the current State as reflected in CurrentActivityStateId was initially entered
ActivityInstantiationTime	DateTime	O	Timestamp of when the Activity was initially made available for the Assignee

1304 **Table 33: ActivityStatusRecord**

1305 *Change History Entries* record information about Activity State Model transitions and Work Request data
1306 updates. Table 34 describes the common abstract base class for different types of Change History
1307 Entries.

ChangeHistoryEntry (abstract base class)			
Property	**Type**	**M/O**	**Description**
ChangeTime	DateTime	M	Timestamp of this change
ResultingRevision	Int	M	Revision number of the Work Request Status Record revision resulting from this change

1308 **Table 34: ChangeHistoryEntry**

1309 *Activity Change History Entries* record information about Activity State Model transitions.

ActivityChangeHistoryEntry (extends ChangeHistoryEntry)			
Property	**Type**	**M/O**	**Description**
AssigneeId	Identifier	O	Identifier of the Assignee who initiated the Action, if not Manager initiated. AssigneeId MUST NOT be specified if Manager initiated.
ActivityId	Identifier	M	Identifier of the associated Activity
StateId	Identifier	M	Identifier of the resulting State after transition
StepId	Identifier	M	Identifier of the resulting Step after transition
ActionId	Identifier	M	Identifier of the Action that caused this transition
InputData	Sequence of DataBinding	O	Input data provided by the Assignee or Manager for the input form of the Action, if an input form is specified for the Action. The order in which bindings are listed is not significant.

1310 **Table 35: ActivityChangeHistoryEntry**

1311 *Data Change History Entries* record changes to Work Request data.

DataChangeHistoryEntry (extends ChangeHistoryEntry)			
Property	**Type**	**M/O**	**Description**
AssigneeId	Identifier	O	Identifier of the Assignee who initiated the Action, if not Manager initiated. AssigneeId MUST NOT be specified if Manager initiated.
Cause	Identifier	M	Either “Action” if this data change was caused by a Data Update Operation associated with an invoked “Action” or “Update” if this data change was caused by Assignee directly updating an updateable Data Element. If Cause is “Action” then the preceding Change History Entry MUST be an Activity Change History Entry describing the Action that caused the update.
UpdatedData	Sequence of DataBinding	M	Updated Work Request data. The order in which bindings are listed is not significant.

Table 36: DataChangeHistoryEntry

8.2.1.10 Work Request Update Structures

Data types specified in this section are used in conjunction with the WR_PUT and WR_INVOKE_ACTION operations to update the content or state of Work Requests via the FFMII interface while managing potential update collisions between the user interface initiated and Manager initiated updates.

The detection and management of update collisions is based on the revision number (RevisionNumber) of the Work Request Status Record associated with the Work Request being updated. The revision number is incremented every time the state of the Work Request changes or the associated data is updated.

When updating the Work Request or invoking an Action on it, the Manager MAY specify the latest known revision number of the associated Work Request Status Record. If specified, the FFMII Implementation MUST checks that this is the latest revision before performing the update or otherwise signals an update collision. If latest revision is not specified, then the Implementation MUST perform the update.

On update collision, the Manager SHOULD read the updated status using WR_GET_STATUS, apply any state changes locally and retry the update operation based on the latest known status, if the operation is still applicable.

WrPutUpdateRequest is used when updating Work Request data content using WR_PUT.

WrPutUpdateRequest			
Property	**Type**	**M/O**	**Description**
WorkRequest	WorkRequest	M	Work Request being added or updated. An update replaces the existing WorkRequest. See notes of WR_PUT operation in Section 8.2.2.
BaseRevisionNumber	Int	O	The revision number of the associated Work Request Status Record this update is based on, or -1 when adding a new non-existing Work Request. If the revision number is specified for Work Request update then the FFMII Implementation MUST check that it matches the latest revision of the associated Work Request Status Record or otherwise signal a WR_UPDATE_COLLISION error. If -1 is specified and a Work Request with the same identifier already exists, the Implementation MUST signal a WR_EXISTS error.

1329 **Table 37: WrPutUpdateRequest**

1330 *WrInvokeActionUpdateRequest* is used to update Work Request status by invoking a specific Action on a
1331 specific Activity of the Work Request.

WrInvokeActionUpdateRequest			
Property	**Type**	**M/O**	**Description**
WorkRequestId	Identifier	M	Identifier of the Work Request to be updated
BaseRevisionNumber	Int	O	The revision number of the associated Work Request Status Record this update is based on. If the base revision number is specified then the FFMII Implementation MUST check that it matches the latest revision of the associated Work Request Status Record or otherwise signal a WR_UPDATE_COLLISION error.
ActivityId	Identifier	M	Identifier of the Activity to be updated
ActionId	Identifier	M	Identifier of the Action to be invoked on the specified Activity

InputData	Sequence of DataBinding	O	Input data to be submitted against the input form of the Action. The provided data is validated according to the validation rules specified in the Data Form, just as user provided input would be. The Implementation MUST validate the input and respond with error codes E3015 or E3016 if the input data is of illegal type or fails validation. An empty Sequence of Data Binding is equivalent to omitting InputData altogether. Input data MUST be omitted if the Action has no input form and MAY be omitted if the input form accepts an empty data set.

1332 **Table 38: WrInvokeActionUpdateRequest**

1333 ### 8.2.1.11 Work Request Query and Response Structures

1334 Data types specified in this section are used in conjunction with the WR_LIST operation used for querying
1335 identifiers of Work Requests managed by the Implementation and accessible to the requesting Manager.

1336 WR_LIST operation, by default, returns identifiers of all Work Requests accessible (visible) to given
1337 Manager. With *WrListFilter* structure provided as input parameter, the content of the response is further
1338 restricted as described in Table 39:

WrListFilter			
Property	**Type**	**M/O**	**Description**
RevisedAfter	DateTime	O	If provided, excludes from the response all Work Requests that have not been modified, or experienced any state change, after specified point in time.
TaskState	Sequence of Identifier	O	Return only Work Requests whose Work Request Status (as defined in 5.1.5) is one of the values contained in TaskState. TaskState is a combination of Identifiers [Open, Active, Inactive, Closed]

1339 **Table 39: WrListFilter**

1340 *WrListResult* encapsulates core identification information about Work Requests.

WrListResult			
Property	**Type**	**M/O**	**Description**
WorkRequestId	Identifier	M	Identifier of the Work Request
CurrentTaskStateId	Identifier	M	Either Open, Active, Inactive, or Closed reflecting the Work Request Status (as defined in 5.1.5)
RevisionNumber	Int	M	Revision number of the associated Work Request Status Record

1341 **Table 40: WrListResult**

1342 8.2.1.12 Work Request Status Change Notification Structures

1343 Data types specified in this section are used together with the WR_NOTIFY_STATUS operation to send
1344 information about status changes of Work Requests.

WorkRequestStatusChangeNotification			
Property	**Type**	**M/O**	**Description**
WorkRequestId	Identifier	M	Identifier of the Work Request
StatusSnapshot	StatusSnapshotRecord	M	Current status of the Task
Changes	Sequence of ChangeHistoryEntry	M	Chronological sequence of one or more new changes to the Work Request since the last successfully delivered notification
CustomProperties	Dictionary	O	Implementation-specific custom properties

1345 **Table 41: WorkRequestStatusChangeNotification**

1346 8.2.1.13 Other Work Request Operation Result Structures

1347 Data types specified in this section are used as result types for various Work Request related batch
1348 operations, as defined in Section 8.2.2.

1349 *WrPutResult* is used as the result type of WR_PUT batch operation. The identifier property of
1350 BatchItemResult identifies the Work Request specified in the request.

WrPutResult (extends BatchItemResult)			
Property	**Type**	**M/O**	**Description**
			Only base class properties

1351 **Table 42: WrPutResult**

1352 *WrGetStatusResult* is used as the result type of WR_GET_STATUS batch operation. The identifier
1353 property of BatchItemResult identifies the Work Request specified in the request.

WrGetStatusResult (extends BatchItemResult)			
Property	**Type**	**M/O**	**Description**
StatusRecord	WorkRequestStatusRecord	O	Status Record of the Work Request identified by the identifier property of this Result. MUST be present if ErrorCode of this result is E0000.

1354 **Table 43: WrGetStatusResult**

1355 *WrInvokeActionResult* is used as the result type of WR_INVOKE_ACTION batch operation. The identifier
1356 property of BatchItemResult identifies the Work Request specified in the request.

WrInvokeActionResult (extends BatchItemResult)			
Property	**Type**	**M/O**	**Description**
			Only base class properties

1357 **Table 44: WrInvokeActionResult**

1358 WrNotifyStatusResult is used as the result type of WR_NOTIFY_STATUS batch operation. The identifier
1359 property of BatchItemResult identifies the Work Request specified in the request.

WrNotifyStatusResult (extends BatchItemResult)			
Property	**Type**	**M/O**	**Description**
			Only base class properties

1360 **Table 45: WrNotifyStatusResult**

1361

1362 8.2.2 Operations

1363 This chapter describes the operations associated with the Work Request Management.

1364

Operation	Description	Notes
WR_PUT [exposed by Implementation]	Creates new Work Requests or updates existing ones with the same identifier. Input parameters: Updates: Sequence of WrPutUpdateRequest Return value: Results: Sequence of WrPutResult	If an existing Work Request is updated: 1. Constraints on changes of associated Work Type Specification are specified in Section 5.1.2.6. 2. The associated Status Record remains as is except that the UpdatedData property

Operation	Description	Notes
		is cleared and the RevisionNumber property is incremented. For details on request structures and how potential update collisions are managed, see Section 8.2.1.10.
WR_LIST [exposed by Implementation]	Returns identifiers and status information for Work Requests matching the specified filter criteria (if provided). Input parameters: Filter: WrListFilter Return value: Results: Sequence of WrListResult	Implicitly restricted to Work Requests visible to the requesting Manager. For detailed description of the filter and result structures, see Section 8.2.1.11.
WR_GET_STATUS [exposed by Implementation]	Retrieves Work Request Status Record for the identified Work Requests. Input parameters: WorkRequestIds: Sequence of Identifier Return value: Results: Sequence of WrGetStatusResult	
WR_INVOKE_ACTION [exposed by Implementation]	Invokes the specified list of Actions on specified Activities and Work Requests, in the order they were specified. Each Action is invoked just like user would have performed it. Input parameters: Updates: Sequence of WrInvokeActionUpdateReque	This operation can be used, for example, to cancel a Work Request or to suspend and resume Activities, provided that corresponding Actions are available in the Activity State Model. The identified Activity MUST be specified the Work Type

Operation	Description	Notes
	st Return value: Results: Sequence of WrInvokeActionResult	Specification of the Work Request and the identified Action in the current Step of the Activity. The provided input data is validated against the input form of the Action. For details on request structures and how potential update collisions are managed, see Section 8.2.1.10.
WR_NOTIFY_STATUS [exposed by Manager]	Delivers information about status changes of one or more Work Requests from Implementation to the Manager when status change notifications are used in contrast to Implementation polling the status with the WR_GET_STATUS operation. Input parameters: Notifications: Sequence of WorkRequestStatusChangeNotification Return value: Results: Sequence of WrNotifyStatusResult	

Table 46: Work Request Management Operations

8.3 Reference Data Management

8.3.1 Introduction

Reference Data Management (RDM) provides means for the Manager to establish custom data repositories with arbitrary content within scope of the Implementation. The content of such repositories is commonly denoted as “Reference Data”, and MAY be used for input value selection, lookup of display values or content validation in Work Requests. A further use is to provide additional information such as documents for Assignees. The objective of the RDM subsystem is to enforce a unified way of managing data content in all repositories, and a common way of referring to repositories and their content across data types, across repositories and from within Work Requests. In order to do so, a common set of generic repository management operations is defined, as well as a common format of unique data type

identifiers and fully qualified references. See Section 5.2 for an overview of the RDM domain model.

An Implementation MAY also provide access to system repositories providing access to selected data on Implementation side, such as Assignee identities and alike. System repositories have reserved identifiers, and they MAY impose restrictions on their content. FFMII interface defines single system repository for managing user information as described in Section 8.9.

8.3.2 Data Types

8.3.2.1 Repository Descriptor

Each repository within RDM is has repository descriptor, structure of which is defined in Table 47:

RepositoryDescriptor			
Property	**Type**	**M/O**	**Description**
Id	Identifier	M	Repository identifier
Description	String	O	Verbal description of the repository
Readable	Boolean	M	For custom repositories, Readable specifies whether other Managers than the creator have permission to read repository content using Reference Data Management services. The manager that created the repository MUST be able to read repository content using Reference Data Management services. For system repositories, semantics is specified as part of the description of the repository.
Writable	Boolean	M	For custom repositories, Writable specifies whether other Managers than the creator have permission to update repository content using Reference Data Management services. The manager that created the repository MUST have full write access (update, initialize, delete) to repository using Reference Data Management services. For system repositories, semantics is specified as part of the description of the repository.
CustomProperties	Dictionary	O	Implementation-specific custom properties

Table 47: Repository Descriptor

8.3.2.1.1 Custom Repository Identifiers

Identifiers of custom repositories MUST begin with prefix “X”. A Manager MUST NOT create a custom repository without the prefix and MUST use system repositories only for the purposes described in the FFMII specification.

8.3.2.2 Reference Data Item

The content of all RDM repositories MUST use the common data type model described in Section 5.2.

Each *Reference Data Item* managed by RDM MUST have an identifier unique within the repository, and MAY contain a *value* and a Sequence of *references to other items*. Any item may therefore act equally as a leaf Reference Data Item, as a collection of references, or as a combination of both.

The Table 48 specifies the structure of a *Reference Data Item*.

ReferenceDataItem			
Property	**Type**	**M/O**	**Description**
Id	Identifier	M	Unique identifier in the context of Reference Data Items stored into a specific repository.
Value	(varies, see below)	O	Value associated with this item
References	Sequence of Reference	O	Further references associated with this item.

Table 48: Reference Data Item

Type of *Values* of Reference Data Items in *custom repositories* MUST be primitive data type (see Section 7.2), Dictionary (see Section 7.4), MLText (see Section 7.5.1), DataAttachmentValue (see Section 8.5.9), DataMatrixValue (see Section 8.5.9), Location (see Section 7.5.2) or MultiChoiceAlternative (see Section 7.5.3). Allowed type of Value in *system repositories* is specific to a particular repository and subject to constraints of the repository (see Sections 8.9, 8.10 and 8.11).

References allow linking with other Reference Data Items managed through RDM. References can be fully-qualified, referring to items in any RDM repository, or relative, referring to items in the same repository.

Reference			
Property	**Type**	**M/O**	**Description**
ItemId	Identifier	M	Identifier of the reference item within its containing repository
RepositoryId	Identifier	O	Identifier of the target repository in case of fully qualified references. If RepositoryId is not specified, the reference is interpreted as a relative reference to the same repository

Table 49: Reference

8.3.2.3 Reference Data Operation Result Structures

Data types specified in this section are used as result types for various Reference Data related batch operations, as defined in Section 8.3.3.

RdInitResult is used as the result type of RD_INIT batch operation. The identifier property of BatchItemResult identifies the Reference Data Item specified in the request.

RdInitResult (extends BatchItemResult)

Property	Type	M/O	Description
			Only base class properties

1411 **Table 50: RdInitREsult**

1412 *RdPutResult* is used as the result type of RD_PUT batch operation. The identifier property of
1413 BatchItemResult identifies the Reference Data Item specified in the request.

RdPutResult (extends BatchItemResult)			
Property	**Type**	**M/O**	**Description**
			Only base class properties

1414 **Table 51: RdPutResult**

1415 *RdGetResult* is used as the result type of RD_GET batch operation. The identifier property of
1416 BatchItemResult identifies the Reference Data Item specified in the request.

RdGetResult (extends BatchItemResult)			
Property	**Type**	**M/O**	**Description**
Item	ReferenceDataItem	O	Reference Data Item identified by the identifier property of this Result. MUST be present if ErrorCode of this result is E0000.

1417 **Table 52: RdGetResult**

1418 8.3.3 Operations

1419 Following Table 53 and Table 54 specify interface operations for manipulation RDM repository content as
1420 well as operations for creation and removal of custom repositories:

1421

Operation	Description	Notes

Operation	Description	Notes
RD_INIT [exposed by Implementation]	Initializes specified repository with the specified set of Reference Data Items. Input parameters: RepositoryId: Identifier Items : Sequence of Reference Data Item Return value: Results: Sequence of RdInitResult	If Repository does not already exist, then RD_INIT creates a new repository with a default Repository Descriptor All existing Reference Data Items, if any, are removed from the repository before new items are stored
RD_PUT [exposed by Implementation]	Inserts or replaces/updates data in a repository. Input parameters: RepositoryId : Identifier Items : Sequence of Reference Data Item Return value: Results: Sequence of RdPutResult	Replaces any existing Reference Data Items that have the same identifier If no Reference Data Item with a specified identifier exists, a new one is added to the repository.

Operation	Description	Notes
RD_DELETE [exposed by Implementation]	Removes identified Reference Data Item(s) from the repository. Input parameters: RepositoryId: Identifier ItemIds: Sequence of Identifier Return value: NumDeleted: Integer	Silently ignores non-existing items Returns the number of items that were deleted
RD_LIST [exposed by Implementation]	Returns a Sequence of Identifiers identifying Reference Data Items in the specified repository. Input parameters: RepositoryId: Identifier Offset: Integer (optional) MaxResults: Integer (optional) Return value: ItemIds: Sequence of Identifier MoreAvailable: Boolean	Starts with the item at the specified Offset (zero based), or with the first item by default. Returns at most MaxResult identifiers or all remaining item identifiers by default. The order in which the item identifiers are returned is implementation-specific but the order MUST be consistent. Returns flag MoreAvailable telling whether there are more items available.

Operation	Description	Notes
RD_GET [exposed by Implementation]	Retrieves the identified or all Reference Data Items contained in the specified Reference Data Repository. Input parameters: RepositoryId: Identifier ItemIds: Sequence of Identifier (Optional) Return value: Results: Sequence of RdGetResult	If ItemIds is specified then retrieves only the identified Reference Data Items contained in the repository. If no item with the specified identifier exists then the Implementation MUST return an error result for the identifier. If ItemIds is not specified then retrieves all Reference Data Items contained in the repository.
RD_REFS_GET [exposed by Implementation]	Retrieves references of the specified Reference Data Item. Input parameters: RepositoryId: Identifier ItemId: Identifier Return value: References: Sequence of Reference	

Operation	Description	Notes
RD_REFS_PUT [exposed by Implementation]	Stores references as part of the specified Reference Data Item. Input parameters: RepositoryId: Identifier ItemId: Identifier References: Sequence of Reference Return value: NONE	Silently ignores existing duplicate references
RD_REFS_DELETE [exposed by Implementation]	Removes all references from the specified Reference Data Item. Input parameters: RepositoryId: Identifier ItemId: Identifier Return value: NONE	

1422 **Table 53: Common Reference Data Operations**

1423 Additionally, the following operations are defined for manipulating the Reference Data repository
1424 catalogue:

1425

Operation	Description	Notes
RD_REPO_LIST [exposed by Implementation]	Retrieves repository descriptors of all repositories available to the requesting Manager. Input parameters: NONE Return value: Repositories: Sequence of RepositoryDescriptor	
RD_REPO_CREATE [exposed by Implementation]	Creates a new empty custom repository. Input parameters: Descriptor: RepositoryDescriptor Return value: NONE	Implicitly sets values of Writable to True, irrespectively of the values supplied in the descriptor parameter (i.e. Manager is always granted access to repositories it creates). Fails if the repository exists already.
RD_REPO_DELETE [exposed by Implementation]	Removes the specified custom repository. Input parameters: RepositoryId: Identifier Return value: NONE	Silently ignores non-existing repositories. Fails if the repository is not empty or is a system repository.

Operation	Description	Notes
RD_REPO_UPDATE_PROPS [exposed by Implementation]	Replaces custom properties in descriptor of the specified custom repository with the specified properties. Input parameters: RepositoryId: Identifier CustomProperties: Dictionary Return value: NONE	Fails if the repository does not exist or is not writable by the Manager

Table 54: Repository Catalogue Management Operations (RDM)

8.4 Field-Initiated Requests

8.4.1 Data Type

This section describes the data types associated with Field-Initiated Requests. The main data types are FieldInitiatedRequestSpecification, FieldInitiatedRequest, and FieldInitiatedRequestResponse. They are specified in dedicated sections together with the additional data types they refer to.

8.4.1.1 Field-Initiated Request Specification

A FieldInitiatedRequestSpecification data type describes how a particular Field-Initiated Request is made available by a Manager to the Implementation. See discussion in Section 5.4.

How Field Initiated Request Specifications are made available in context of each Work Request is described in the Work Type Specification. Several Work Requests may share the same Field-Initiated Request Specification by referring to them.

FieldInitiatedRequestSpecification			
Property	**Type**	**M/O**	**Description**
FIRType	Identifier	M	Either "TopicalInquiry" or "TopicalNotification"
TopicLabel	MLText	M	Used for visualization purposes by the Implementation
GroupLabel	MLText	O	Used for visualizing grouping of different Topics
SharedDataElements	Sequence of DataElementSpecification	O	Shared Data Element Specifications that may be referred to from within Data Forms specified in this Field Initiated Request Specification.
RequestForm	DataForm	O	Specifies data to be supplied by the Assignee
ResponseForm	DataForm	O	Specifies data to be returned by Manager, if FIRType is “TopicalInquiry”. ResponseForm MUST NOT be specified if FIRType is “TopicalNotification”.
ReturnsWRs	Boolean	M	Whether Manager returns Work Requests as a response. MUST be False if FIRType is “TopicalNotification”.
AvailableTopicsForReturnedWRs	Sequence of Identifier	O	Set of available Work Request processing Topics, if Manager returns Work Requests (Manager identifier is implicitly the identifier of the Manager returning the Work Requests)
BoundToWR	Boolean	M	True if resulting FIR must be bound to a Work Request

1439 **Table 55: FieldInitiatedRequestSpecification**

1440 ### 8.4.1.2 Field-Initiated Request

1441 A FIR data type embodies a Field-Initiated Request. See discussion in Section 5.4.

FieldInitiatedRequest			
Property	**Type**	**M/O**	**Description**
Id	Identifier	M	Unique Identifier generated by the Implementation
Timestamp	DateTime	M	Time of request initiation
TopicId	Identifier	M	Identifies the requested Topic (no need to specify Manager Identifier since this is sent to a specific Manager)
AssigneeId	Identifier	M	Identifier or the initiating Assignee
WorkRequestId	Identifier	O	identifier of the Invocation context (Work Request, Activity and Step) if Field-Initiated Request was invoked in the context of a specific Work Request
ActivityId	Identifier	O	
StepId	Identifier	O	
RequestData	Sequence of DataBinding	M	Request data supplied by the Assignee (may be empty if allowed by corresponding Request Form specification)

Table 56: FieldInitiatedRequest

8.4.1.3 Field-Initiated Request Response

A Manager MUST respond to a Field-Initiated Request of FIRType TopicalInquiry with one or more Field-Initiated Request Response. See discussion in Section 5.4.

FieldInitiatedRequestResponse			
Property	**Type**	**M/O**	**Description**
FieldInitiatedRequestId	Identifier	M	identifier of the associated Field-Initiated Request
SeqNum	Integer	M	Starting with 0 and monotonically increasing for further responses associated with the same Field-Initiated Request
Timestamp	DateTime	M	Time of response
ResponseData	Sequence of DataBinding	M	Data retuned by Manager (may be empty)
WorkRequests	Sequence of WorkRequests	O	Any number of Work Requests returned by the Manager, if allowed by the specification associated with the request
FinalIndicator	Boolean	M	True if response is final, False if this Field-Initiated Request Response is intermediate. An intermediate response MUST be followed by a later Field-Initiated Request Response. A later Field-Initiated Request Response MUST override any information contained in previous Field-Initiated Request Responses to the same request.

Table 57: FieldInitiatedRequestResponse

8.4.1.4 Field-Initiated Request Operation Result Structures

Data types specified in this section are used as result types for various Field-Initiated Request related batch operations, as defined in Section 8.4.2.

FirGetResponseResult is used as the result type of FIR_GET_RESPONSE batch operation. The identifier property of BatchItemResult identifies the Field-Initiated Request specified in the request.

FirGetResponseResult (extends BatchItemResult)			
Property	**Type**	**M/O**	**Description**
Responses	Sequence of FieldInitiatedRequestResponse	O	Responses to Field-Initiated Request identified by the identifier property of this Result. Responses MUST be in chronological order. MUST be present if ErrorCode of this result is E0000. The sequence MUST be present but empty if Field-Initiated Request is known and accessible to the Implementation but no new responses are available for it.

Table 58: FirGetResponseResult

FirNotifyResponseResult is used as the result type of FIR_NOTIFY_RESPONSE batch operation. The identifier property of BatchItemResult is the Field-Initiated Request identifier of the response being acknowledged by this result.

FirNotifyResponseResult (extends BatchItemResult)			
Property	**Type**	**M/O**	**Description**
SeqNum	Integer	M	SeqNum of the Field-Initiated Request Response being acknowledged by this result.

Table 59: FirNotifyResponseResult

8.4.2 Operations

The Field-Initiated Requests subsystem exposes the following operations:

Operation	Description	Notes
FIR_EXECUTE [exposed by Manager]	Implementation uses this operation to send a Field-Initiated Request to Manager. Input parameters: FieldInitiatedRequest: FieldInitiatedRequest Return value: NONE	
FIR_GET_RESPONSE [exposed by Manager]	Implementation uses this operation to check if the Manager has generated responses for specified Field-Initiated Requests sent by Implementation. The Implementation specifies the Field-Initiated Request for which responses are requested by the identifier of the Field-Initiated Request. Input parameters: FieldInitiatedRequestIds: Sequence of Identifier Return value: Results: Sequence of FirGetResponseResult	The returned sequence may be empty if no responses are available yet. If the results include more than one Field-Initiated Request Response data type, the Manager may choose to provide all of them in one operation, or provide a smaller number of Field-Initiated Request Responses in each response, Implementation SHOULD continue polling until it receives a response with the FinalIndicator field set to True. If the requested Field-Initiated Request identifier does not specify a Field-Initiated Request which the Manager is currently processing, Manager MUST return an UNKNOWN_FIR_ID error. This polling mechanism is intended for use when both Manager and Implementation are configured for using polling in Field-Initiated Request processing. The configuration mechanism is not specified by this standard.

Operation	Description	Notes
FIR_NOTIFY_RESPONSE [exposed by Implementation]	Manager uses this operation to send FIR results to the requesting Implementation. Input parameters: Responses: Sequence of FieldInitiatedRequestResponse Return value: Results: Sequence of FirNotifyResponseResult	If the results include more than one Field-Initiated Request Response data type, a Manager may choose to provide all of them in one operation, or provide a smaller number of Field-Initiated Request Responses in each response. An Implementation SHOULD expect more responses until it receives a response with the FinalIndicator field set to True. This notification mechanism is intended for use when both Manager and Implementation are configured for using notification in Field-Initiated Request processing. The configuration mechanism is not specified by this standard.

1459 **Table 60: Field-Initiated Request Operations**

1460 # 8.5 Data Form Data Types

1461 ## 8.5.1 Data Form and Data Elements

1462 This section describes the Data Form class together with core Data Element classes. Data Element
1463 Specification subclasses are described in following sections.

1464 A Data Form aggregates a sequence of Data Elements that form a data view or an input form. The order
1465 of Data Element declarations is significant and the Implementation SHOULD use it as a hint for ordering
1466 the elements within the user interface.

DataForm			
Property	**Type**	**M/O**	**Description**
Elements	Sequence of DataElement	M	One or more Data Elements

1467 **Table 61: DataForm**

1468 A Data Element describes a named piece of data or a data group. They provide the information needed
1469 by the user interface implementation to display the data fields together with the desired label and
1470 grouping. A value can be bound to a Data Element via a Data Binding included in the associated data
1471 type providing the instance data such as Work Request, Field-Initiated Request or Field-Initiated Request
1472 Response.

1473 Data Elements MAY contain validation rules. User provided data MUST be validated against the specified
1474 validation rules.

1475 Data Elements are defined either as in-lined Data Element Specifications, specifying the details of the
1476 Data Element, or as Data Element References, referring to one of the shared Data Element
1477 Specifications specified in the SharedDataElements property of the Work Type Specification or Field-
1478 Initiated Request Specification contains the reference.

DataElement (abstract base class)

Property	Type	M/O	Description
			No Common properties

1479 **Table 62: DataElement (abstract base class)**

1480

DataElementReference (extends DataElement)			
Property	**Type**	**M/O**	**Description**
Id	Identifier	M	Identifies a shared Data Element Specification to be substituted in place of this reference. There MUST exist a Data Element Specification with the specified identifier in the SharedDataElements property of the Work Type Specification or Field Initiated Request Specification containing this reference.

1481 **Table 63: DataElementReference (extends DataElement)**

1482 There are several subclasses of Data Element Specifications. All subclasses have the following common
1483 properties as part of the common abstract base:

DataElementSpecification (abstract, extends DataElement)			
Property	**Type**	**M/O**	**Description**
Id	Identifier	M	Unique identifier among the Data Element Specifications specified in the containing Work Type Specification or Field-Initiated Request Specification.
Label	MLText	M	Label for user interface purposes. Data variables (see section 8.5.12) MUST be Supported in Labels.
Keyword	MLText	O	Short keyword or mnemonic to be used in a space-limited user interface, for example SMS
HelpText	MLText	O	Help text describing the content of the element. . Data variables (see section 8.5.12) MUST be Supported in HelpText,
ValidationCondition	Expression	O	Validation condition specifies an Expression that evaluates to True for valid values. The Implementation MUST verify that any user provided data satisfies the ValidationCondition. If not present, any value is considered as valid. See section 8.5.6 for details.
EnableCondition	Expression	O	Condition that defines when this Data Element is enabled. If the element is not enabled then it is not shown nor is it validated. If not present, the Data Element is considered

DataElementSpecification (abstract, extends DataElement)			
Property	**Type**	**M/O**	**Description**
			as enabled. See section 8.5.7 for details.
UpdateableCondition	Expression	O	Specifies whether the value of this Data Element may be updated by the user. Default is False for Data Elements in data views and True for Data Elements in input forms. This property makes it possible to introduce user-updateable Data Elements in data views, for example to make it possible to update incorrect background data associated with the work.
Formatting	Sequence of Identifier	O	Sequence of predefined or implementation-specific formatting tags that further specify how the data should be formatted. The order in which the formatting tags are listed is not significant. See section 8.5.8 for details.
Source	Identifier	O	A predefined or implementation-specific tag providing a hint on how the content should be obtained. An Implementation SHOULD try to comply with this tag. See section 8.5.7 for details.

1484 **Table 64: DataElementSpecification (abstract, extends DataElement)**

1485 8.5.2 Data Field Specification

1486 Data Field Specification specifies a simple data field.

1487 Multiple choice fields can be created by specifying valid MultiChoiceAlternative values. It is up to the user
1488 interface implementation to decide how to present the data field, e.g. depending on the number of the
1489 possible values.

DataFieldSpecification (extends DataElementSpecification)			
Property	**Type**	**M/O**	**Description**
Type	Identifier	M	Type of the data field value. The supported values are names of primitive data types (see section 7.2), and “Location” for Location data type (see Section 7.5.2)
UnitLabel	MLText	O	Unit label to be displayed for this data field (e.g. “€”, “km”, “days”, etc). Data variables (see section 8.5.12) MUST be Supported in Unit label.
Alternatives	Sequence of MultiChoiceAlternative	O	Sequence of valid values for this multiple choice fields. Specified value MUST be compatible with this Data Field Specification. If Alternatives is specified then the Implementation MUST NOT accept any other values for this field. The Implementation SHOULD display the alternatives to the Assignee in the order they were specified. This property MUST NOT be specified if the AlternativesRespositoryId property is specified.
AlternativesRepositoryId	Identifier	O	Identifier of a custom Reference Data repository that specifies the valid values for this multiple choice field as values of the contained Reference Data Items. The values of Reference Data Items MUST be either of type MultiChoiceAlternative containing an optional label together with the value or a plain value. Specified values MUST be compatible with this Data Field Specification. If AlternativesRepositoryId is specified then the Implementation MUST NOT accept any other values for this field. The Implementation SHOULD assist the Assignee in choosing one of the valid values. The Reference Data MAY also form a selection tree structure, as described in Section 8.5.2.1. This property MUST NOT be specified if the Alternatives property is specified.
PrimaryAlternatives	Sequence of MultiChoiceAlternative	O	Provides a sequence of most likely input values for this field, in the decreasing order of importance. This property is only meaningful for input fields or updateable output fields.

DataFieldSpecification (extends DataElementSpecification)			
Property	**Type**	**M/O**	**Description**
			The Implementation MAY use this property to offer the user shortcuts to most likely input values, while also allowing other valid values to be entered. The user interface should generally display the alternatives in the order they were specified, unless further heuristics are available. The intention is to enhance usability, especially in compact mobile user interfaces. Any specified alternatives that do not satisfy the validation conditions of this field MUST be ignored.

1490 **Table 65: DataFieldSpecification (extends DataElementSpecification)**

1491 ### 8.5.2.1 Hierarchical Selection Tree

1492 A Data Field Specification MAY specify a hierarchical selection tree to assist the Assignee in selecting a
1493 correct value for the field when there is a large set of valid alternatives and they can be organized in a
1494 hierarchical tree structure. For example, replacement part codes could be organized into a selection tree
1495 where the first level would be a device category, the second level a device type, the third level a device
1496 model, and the fourth level a replacement part code.

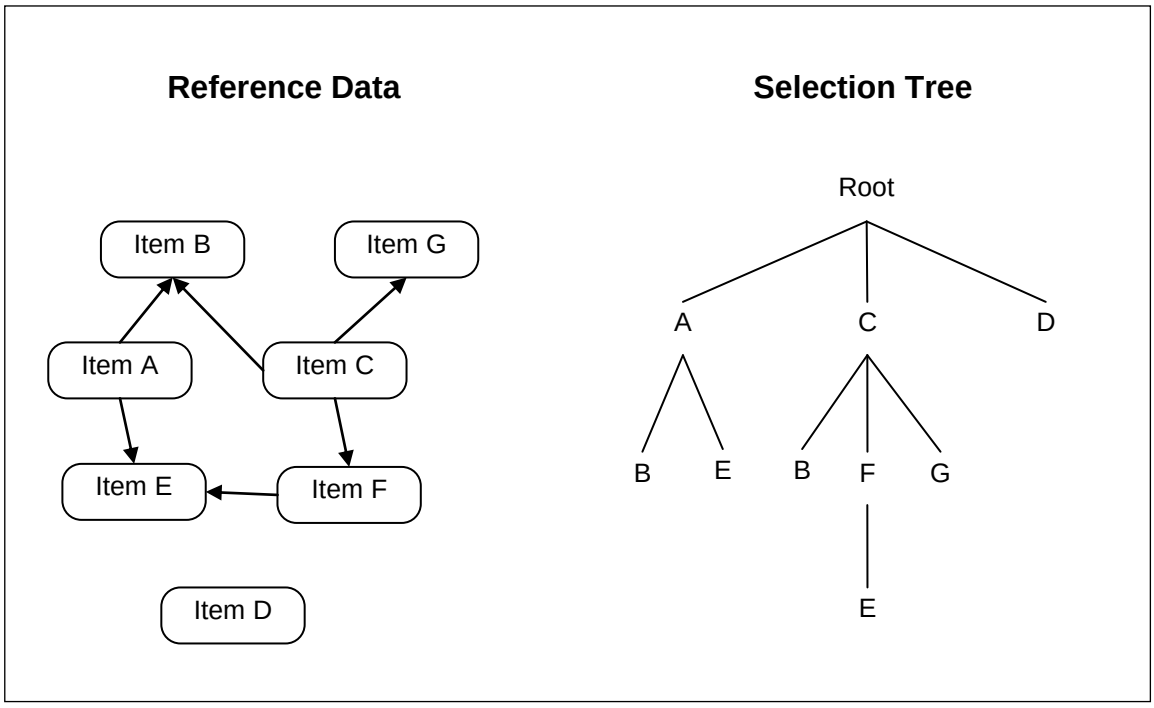
1497 A hierarchical selection tree can be specified by storing all valid alternatives and their hierarchy into a
1498 custom Reference Data repository that is identified in property AlternativesRepositoryId of the Data Field
1499 Specification. In this case the references between Reference Data Items stored into the repository define
1500 the hierarchy as follows.

1501 Those Reference Data Items in the identified repository that are not referred to from any other items in
1502 the repository form the top level of the selection tree, i.e. the first set of selections presented to the
1503 Assignee. The Reference Data Items referred to from each top level item form the next level of the
1504 selection tree, and so forth. The Reference Data Items not referring to any other items are the leaf nodes
1505 of the selection tree. The Assignee is presented with selections starting from the top level selections and
1506 narrowing down the value set until a leaf node is selected, determining the chosen value.

1507 A Reference Data Item stored in the identified repository MAY refer to an item stored in another
1508 repository. A Reference Data Item that is part of a selection tree MUST NOT contain a cyclic reference to
1509 itself either directly or indirectly via an upper level item. A single Reference Data Item MAY be referred to
1510 from more than one other item, i.e. it may belong to several sub-trees.

1511 Each Reference Data Item being part of a selection tree MUST have a Value. If the type of Value is
1512 MultiChoiceAlternative then its Label, if any, is used as the display label of the selection and Value as the
1513 selection value. Otherwise the plain Value is used as the display label of the selection. The
1514 Implementation MUST NOT accept values other than those specified by the leaf nodes of the tree for the
1515 field.

1516 As an example (see Figure 25), let there be Reference Data Items from A to G stored into a custom
1517 Reference Data repository used as the source of alternatives for a field. Item A refers to Items B and E.
1518 Item C refers to Items B, F and G. Item F refers to E. Items A, C, and D are top level selections and Items
1519 B, D, E, G are leaf-nodes.



1520

1521 **Figure 25: Hierarchical Selection Tree - Example**

1522 ### 8.5.3 Data Attachment Specification

1523 Data Attachment Specification specifies an attachment type of a Data Element. The exact attachment
1524 type, file name and content is provided as Data Attachment Value (see Section 8.5.9 for details).

DataAttachmentSpecification (extends DataElementSpecification)			
Property	**Type**	**M/O**	**Description**
MimeTypeBase	String	O	Base MIME type requirement for the attachment in accordance with [RFC2046]. For example, “image” specifies that an attachment should be an image. If no base type has been specified then any attachment is valid. The base type can also be used as a hint for the user interface implementation on how to show or capture the related input data. An attachment of type “image” could be shown as an image and in an input form it might enable photo shooting, if such feature is available.
MaxSize	Int	O	Maximum allowed size of the attachment data in bytes. Default is unlimited, subject to implementation-specific limits.

1525 **Table 66: DataAttachmentSpecification (extends DataElementSpecification)**

8.5.4 Data Matrix Specification

Data Matrix Specification specifies a two-dimensional array of data. The specification includes a data type specification for each column of the array. The array may have any number of rows with each element on a row matching the type specified for the corresponding column. The labels of the column specifications are used as the column labels for user interface purposes.

For example, MatrixElementSpecification with column element types of (String, Int) will specify a table with any number of lines having a string value in the first column and an integer value in the second column.

The UpdateableCondition property inherited from Data Element Specification determines whether user may add new rows to the matrix. It also affects whether user may delete rows and update cell values but this can be overridden by more specific properties in the following Data Matrix related data type specifications.

DataMatrixSpecification (extends DataElementSpecification)			
Property	**Type**	**M/O**	**Description**
Columns	Sequence of DataMatrixColumnSpecification	M	Specifies the labels and element types for the columns of the matrix.
RowsDeletableCondition	Expression	O	Whether the user may delete rows of this matrix. May be overridden by row-specific property Deletable in DataMatrixRowValue. This value is also used as a default value of Deletable property for newly added rows. If not specified, the default value is given by the UpdateableCondition property of this matrix specification.

Table 67: DataMatrixSpecification (extends DataElementSpecification)

Matrix column element types are specified as a Sequence of Data Matrix Column Specifications. It extends Data Field Specification by adding matrix column specific properties.

DataMatrixColumnSpecification (extends DataFieldSpecification)			
Property	**Type**	**M/O**	**Description**
ValueForAddedRow	DataValue	O	If specified then the given value is used automatically for the corresponding column on newly added rows. The user is not asked to enter a value for this column while adding a new row. The type of the specified value must match the Type property. MLText values can be specified for “String” type of fields. Data variables (see Section 8.5.10) MUST be Supported in String and MLText values. MLText value is converted to String value according to the current user locale and any data variables are expanded upon adding the row. To automatically use undefined value for the column, specify a DataValue container with no Value property. If not specified the user is expected to enter a value while adding a new row.

Table 68: DataMatrixColumnSpecification (extends DataFieldSpecification)

8.5.5 Data Group Specification

Data Group Specification specifies a group of Data Elements. It can be used for user interface purposes to group related Data Elements but it does not have any relations to the data content.

DataGroupSpecification (extends DataElementSpecification)			
Property	**Type**	**M/O**	**Description**
Contents	Sequence of DataElement	M	One or more Data Elements contained within this group

Table 69: DataGroupSpecification (extends DataElementSpecification)

8.5.6 Data Element Formatting Tags

The following predefined Data Element formatting tags MAY be used in Data Element Specification (see Section 8.5.11). An Implementation MAY use this information to determine how to display the data. Implementation-specific tags MAY be defined and MUST begin with “X-”.

Formatting Tag	Description
Title	Value of the associated Data Element is to be displayed as the title of a Work Request (actual visual interpretation of this tag is implementation specific). Note that this tag may appear in several forms, typically in Header and Overview of Work Request, in which case it is expected that the visualization paradigm used is consistent across all of them. This formatting tag MUST NOT be used for updatable elements
Subtitle	Like “title”, but visually expressed as an element related to, and further refining the content of element tagged with “title” This formatting tag MUST NOT be used for updatable elements
Header	The column corresponding to this Data Element SHOULD be visualized as a header in a data matrix
Monospace	The value of the element SHOULD be displayed with a fixed width font
Preformatted	The line breaks, if any, present in the value of the element SHOULD be respected and displayed
Phonenumber	The value of the element is a phone number and, if supported by the device, MAY be used for communication services
Weblink	The value of the element is a web URL and MAY be displayed as a hyper link
Log	The associated data matrix SHOULD be displayed as a log of events or data. E.g. usually an ordered list of rows with one column with a key such as a time-stamp, and associated data in other columns.
Collapsed	The Data element SHOULD be displayed initially in a collapsed view, for example displaying only the label and making it possible for the user to expand the element to show its content
Preview	Preview of the Data Element content (for example, first few lines of text or a thumbnail image) SHOULD be displayed but rest of the content MAY be collapsed if the full value would take considerable space on the display. It MUST be possible for the user to expand a collapsed value as needed
Slider	The value of this element MAY be displayed and adjusted along a finite range along an axis, using a user-interface component such as a slider
Icon	The value of this element is an Attachment whose content is a binary representation of a graphic icon, and the element MAY be displayed by displaying that icon

Location	The value of the element specifies a geographical location (see Section 7.5.2) and MAY be associated with navigation and mapping

Table 70: Data Element Formatting Tags

8.5.7 Data Element Source Tags

The following predefined Data Element source tags can be used in Data Elements Specification (see Section 8.5.1). An Implementation MAY use this information to determine how to obtain the data. Implementation-specific tags MAY be defined and MUST begin with "X-".

Source Tag	Description
Camera	Element data SHOULD be obtained using on-device camera and MUST be stored as an Attachment
Barcode	Element Attachment data SHOULD be obtained using on-device or connected bar code reader
GPS	Element data SHOULD be obtained from the on-device GPS
File	Element data SHOULD be obtained from a user selected file (default source) and therefore the element MUST be an Attachment
Signature	Element data SHOULD be obtained using digital signature capture and MUST be stored as an Attachment.

Table 71: Data Element Source Tags

8.5.8 Data Binding

A Data Binding provides a value for a Data Element (see Section 8.5.1) or a Data Variable (see Section 8.5.10).

The WorkRequestData property of a Work Request contains a Sequence of Data Bindings providing initial values for the Data Elements specified in the associated Work Type Specification. Similarly, the RequestData property of a Field Initiated Request and the ResponseData property of a Field Initiated Request Response contain a Sequence of Data Bindings providing values for the Data Elements specified in the RequestForm and ResponseForm properties of the associated Field Initiated Request Specification, respectively.

A Work Request Status Record and the associated change history entries, on the other hand, use Data Bindings to record user provided input and user updated values.

Data Bindings in the WorkRequestData property of a Work Request are also used to provide values for Data Variables (see Section 8.5.9). The same Data Binding can provide a value both for a Data Element and a Data Variable.

A Sequence of Data Bindings sent by the Manager MAY contain one or more Data Bindings with the same identifier as long as they have different values of the Locale property. In this case the receiving Implementation SHOULD use the Value of the Data Binding having the Locale value that is a best match for the locale preference of the associated Assignee.

DataBinding			
Property	**Type**	**M/O**	**Description**
Id	Identifier	M	An identifier of a Data Element specified in the

DataBinding			
Property	**Type**	**M/O**	**Description**
			associated Work Type Specification or Field-Initiated Request Specification or an identifier of a Data Variable to be set to the specified Value
Value	DataValue	O	Value to be set to the identified Data Element or Data Variable. If value is to be set to a Data Element then the type of Value MUST be compatible with the Data Element Specification. If the type of Value is not compatible then the Implementation MUST return an error code E3015. See Section 8.5.9 for details An alternative way of specifying a value is to specify a Value Reference, see below. Either Value or ValueReference MUST be specified but not both.
ValueReference	Reference	O	A Value Reference MAY be specified instead of a Value to refer to a value stored in a Reference Data repository. If Value Reference is specified then the Implementation MUST use the value of the Reference Data Item referred to by the Value Reference as if it was included in the DataValue container of the Value property. If no such Reference Data Item exists then the Implementation MUST behave as if no Value was specified in the DataValue container. The Implementation MUST NOT use Value Reference when communicating user provided input to the Manager as part of Work Request Status Record or Field Initiated Request, even if the user provided input is based on alternatives specified in a custom Reference Data repository. Either Value or ValueReference MUST be specified but not both.
Locale	LocaleSpecifier	O	Specifies the locale for this value.

1574 **Table 72: DataBinding**

1575 ## 8.5.9 Data Values

1576 Data Value is a container for a Data Element value. The type of value must match the type of the Data
1577 Element.

DataValue			
Property	**Type**	**M/O**	**Description**
(name of contained type)	Any primitive type or MLText or DataAttachmentValue or DataMatrixValue or Location	O	A single value of any primitive type or MLText or DataAttachmentValue or DataMatrixValue Not specifying a value means that no value is bound to the associated element (i.e. it's value is undefined).

1578 **Table 73: DataValue**

1579 Data Attachment Value provides a value for Data Attachment Specification.

DataAttachmentValue			
Property	**Type**	**M/O**	**Description**
MimeType	String	O	MIME type of the attachment, for example “image/jpeg”. The type MUST be compatible with the base type specified in the Data Attachment Specification, if any. The correct MIME type SHOULD be specified if it is known. If type is not specified or recognized then the Implementation typically handles the attachment data as opaque binary data.
FileName	String	O	File name for the attachment. A name SHOULD be specified if the attachment is based on a physical file.
Data	Binary	M	Binary data of the attachment.

1580 **Table 74: DataAttachmentValue**

1581 Data Matrix Value provides a value for a Data Matrix Specification. A sequence of Data Matrix Row Value
1582 is included, corresponding to individual rows of the matrix. Each Data Matrix Row Value contains values
1583 for columns of the row, corresponding to the Data Matrix Column Specifications in the Columns property
1584 of Data Matrix Specification.

DataMatrixValue			
Property	**Type**	**M/O**	**Description**
RowValues	Sequence of DataMatrixRowValue	M	Data rows for this matrix

1585 **Table 75: DataMatrixValue**

DataMatrixRowValue			
Property	**Type**	**M/O**	**Description**
Id	Identifier	O	Optional identifier uniquely identifying this row

DataMatrixRowValue			
Property	**Type**	**M/O**	**Description**
			within the matrix. Not used for display purposes. The Implementation MUST return any identifier originally supplied by the Manager. For a row added via the user interface the Implementation MUST generate an identifier starting with prefix “UI-” that uniquely identifies the row among any other rows added to the same matrix via the user interface.
ColumnValues	Sequence of DataValue	M	Provides data for one row of the DataMatrix. Contains one value for each column in the order the columns were defined in the associated Data Matrix specification. The types of values MUST match the corresponding column specifications.
Deletable	Boolean	O	Whether this row may be deleted by the user. The default value is given by the RowsDeletableCondition property of the associated DataMatrixSpecification.
UpdateableCells	Sequence of Boolean	O	A sequence of Boolean values, corresponding to the data cells of this row. The sequence MUST have one value for each matrix column. The values indicate whether the value of the corresponding cell may be updated by the user. The default values are given by the UpdateableCondition property of the corresponding column specification, or if not specified, the Updateable property of associated DataMatrixSpecification.

Table 76: DataMatrixRowValue

8.5.10 Data Variables

Data Variables provide a mechanism for substituting variable values into textual data content. A Data Variable reference occurring in a String type of value is substituted with the value of the Data Variable whenever the String value is evaluated and used.

The Manager MAY refer to Data Variables from within String or MLText type of Values of Data Bindings included in the WorkRequestData property of a Work Request. Additionally, the Manager MAY refer to Data Variables from within String or MLText type of properties of data types only if the description of the property explicitly allows the use of Data Variables. The Manager MUST NOT refer to Data Variables from other Strings.

The Implementation MUST support Data Variables.

A Data Variable reference MUST use the following format (without the enclosing quotes):

- “${variable.name}” or

- "${variable.name/member.name}"

Where, "variable.name" is the identifier of the referred Data Variable. The reference in whole is replaced with the textual representation of the Data Variable value.

Members of composite type variables, such as rows and cells of Data Matrix, can be referred to using "/" (slash sign) as separator between variable and member identifiers. For example, "variable.name/row.name" refers to row identified as "row.name", if such row exists. Correspondingly, "variable.name/row.name/column.name" refers to the cell in the column "column.name" on that row.

Values of Data Variables are specified using Data Bindings (see Section 8.5.8). It is an error if no value is bound to a Data Variable referenced from within a String to be evaluated.

System-defined variables:

In addition to Data Variables bound by Data Bindings, the following system variables are always available and automatically bound to valid, current value by the Implementation:

Variable	Description	Notes
System.CurrentTime	The current date and time	Bound to a value of type DateTime

Table 77: System-defined Variables

8.6 Expressions

Expressions make it possible to declare flexible constructs that can be dynamically evaluated to a value based on Data Element or Data Variable values, current State of an Activity contained in the associated Work Type Specification, reference data, constants or other data that can be referenced. They are used for specifying data validation, visibility and updateability conditions, and enabling conditions for Activities, Actions and Field-Initiated Requests associated with a Step in Work Requests. An Expression can be evaluated to a value but MUST NOT have any side effects. The ability to describe expressions as structured data eliminates the need to implement lexers and parsers to interpret and evaluate expressions.

Expression is an abstract base class for all expressions.

Expression (abstract base class)			
Property	**Type**	**M/O**	**Description**
			No common properties

Table 78: Expression

Constant Expression evaluates into a constant value. If Value property of the specified DataValue container is omitted then the expression evaluates into undefined value.

8.6.1 Constants and Data Access

ConstantExpression (extends Expression)			
Property	**Type**	**M/O**	**Description**
Value	DataValue	M	The constant value this expression evaluates into

1626 **Table 79: ConstantExpression**

1627 *VariableExpression* evaluates into the value bound to the identified Data Element or Data Variable. If no
1628 such Data Element or Data Variable exists, the expression evaluates into undefined value.

1629

VariableExpression (extends Expression)			
Property	**Type**	**M/O**	**Description**
VariableId	String	M	Identifier of a Data Element or a Data Variable in the associated Work Request, Field-Initiated Request or Field-Initiated Request Response. MAY also refer to a member value of a composite type using the same notation as used in Data Variable references. See section 8.5.10 for notation used to address members of composite data types.

1630 **Table 80: VariableExpression**

1631 Unary Operator Expression applies an unary operator on the evaluated value of the parameter
1632 expression. The resulting value depends on the operator and its semantics.

1633 ## 8.6.2 Unary Operators

UnaryOperatorExpression (extends Expression)			
Property	**Type**	**M/O**	**Description**
Operator	Identifier	M	Identifier of the operator to be applied to parameters. Table 82 specifies all valid unary operators. It is an error to specify any other value.
Param	Expression	M	Expression that evaluates to the parameter value

1634 **Table 81: UnaryOperatorExpression**

1635 The following table lists all valid unary operators and their semantics for Unary Operator Expression.

1636

Operator Identifier	Semantics
Defined	Evaluates into Boolean. True when the parameter expression evaluates into some defined value and False when it evaluates into undefined value.
Negation	Evaluates into logical negation of the parameter value. It is an error if the parameter expression does not evaluate into Boolean.

Length	Evaluates into Int. For String parameter returns the number of characters in the string. For DataMatrixValue parameter returns the number of rows in the matrix. It is an error if the type of the parameter value is not String or DataMatrixValue.

1637 **Table 82: Unary Operators**

1638 8.6.3 Binary Operators

1639 Binary Operator Expression applies a binary operator on the evaluated values of left-hand side and right-
1640 hand side parameter expressions. The resulting value depends on the operator and its semantics.

1641

BinaryOperatorExpression (extends Expression)			
Property	**Type**	**M/O**	**Description**
Operator	Identifier	M	Identifier of the operator to be applied to parameters. Table 84 specifies all valid binary operators. It is an error to specify any other value.
LeftParam	Expression	M	Left-hand side parameter expression
RightParam	Expression	M	Right-hand side parameter expression

1642 **Table 83: BinaryOperatorExpression**

1643 The following table lists all valid binary operators and their semantics for Binary Operator Expression.

1644

Operator Identifier	**Semantics**
Less	Evaluates into Boolean. For two numeric parameters both of the same numeric type Int, Double or Decimal returns whether the left-hand side parameter value is numerically strictly less than the right-hand side parameter value. For two parameters both of the same type DateTime, Date or Time returns whether the left-hand side timestamp comes strictly before the right-hand side timestamp. For two parameters of type Duration returns whether the left-hand side duration is strictly less than the right-hand side duration. It is an error if parameters are of some other type or undefined.
LessOrEqual	Evaluates into Boolean. For two numeric parameters both of the same numeric type Int, Double or Decimal returns whether the left-hand side parameter value is numerically

	less than or equal to the right-hand side parameter value. For two parameters both of the same type DateTime, Date or Time returns whether the left-hand side timestamp comes before or is equal to the right-hand side timestamp. For two parameters of type Duration returns whether the left-hand side duration is less than or equal to the right-hand side duration. It is an error if parameters are of some other type or undefined.
Equal	Evaluates into Boolean. For two numeric parameters both of the same numeric type Int, Double or Decimal returns whether the parameter values are numerically equal. For two parameters of type String returns whether the parameter values are identical. For two parameters both of the same type DateTime, Date or Time returns whether the timestamps are equal. For two parameters of type Duration returns whether the durations are equal. It is an error if parameters are of some other type or undefined.
GreaterOrEqual	Identical to “LessOrEqual” but with the order of parameters reversed.
Greater	Identical to “Less” but with the order of parameters reversed.
MatchesPattern	Evaluates into Boolean. Type of both parameter values must be String. Returns True if the left-hand side String completely matches the right-hand side pattern String, and False otherwise. The pattern is a simplified regular expression using the following constructs. • x – matches character x • [abc] – matches one of the characters a, b or c • [a-cf-z] – matches one of the characters a-c or f-z • . - matches any one character • X* - matches expression X zero or more times • X? - matches expression X zero or one time • (,), [,], *, ?, {, }, \, ^, $ - reserved characters for current or future FFMII use • \ - quote next character and treat it as literal
Addition	Evaluates into an arithmetic sum of the parameters (addition operation). For two parameters both of the same type Integer, Double, Decimal or Duration evaluates into a sum of the same type.

	For left-hand side parameter being a timestamp of type DateTime, Date or Time and right-hand side parameter of type Duration evaluates into a timestamp value having the duration added. Type of result value is the same as that of the left-hand side parameter. It is an error if parameters are of some other type or undefined.
Subtraction	Evaluates into an arithmetic difference of the parameters (subtraction operation). For two parameters both of the same type Integer, Double, Decimal or Duration evaluates into a difference of the same type. For two parameters both of the same type DateTime, Date or Time evaluates into a difference of type Duration. For left-hand side parameter being a timestamp of type DateTime, Date or Time and right-hand side parameter of type Duration evaluates into a timestamp value having the duration subtracted. Type of result value is the same as that of the left-hand side parameter. It is an error if parameters are of some other type or undefined.
Multiplication	Evaluates into an arithmetic product of the parameters (multiplication operation). For two parameters both of the same type Integer, Double or Decimal evaluates into a product of the same type. It is an error if parameters are of some other type or undefined.
Division	Evaluates into an arithmetic quotient of the parameters (division operation). For two parameters both of the same type Integer, Double or Decimal evaluates into a quotient of the same type. It is an error if parameters are of some other type or undefined or if the right-hand side parameter is zero.
AerialDistanceBetween	Evaluates to a Double value specifying the aerial distance between the specified Locations in meters, using spherical geometry. Both parameters MUST be of type Location.

1645 **Table 84: Binary Operators**

1646 8.6.4 Conjunction and Disjunction

1647 Conjunction is a Boolean expression that evaluates to True if and only if all of its parameter expressions
1648 evaluate to True. Otherwise it evaluates to False. Parameters of Conjunction are evaluated in the order
1649 they are listed and only up to the first parameter that evaluates to False. It is an error if some parameter
1650 evaluates to a value other than Boolean.

1651

Conjunction (extends Expression)			
Property	**Type**	**M/O**	**Description**
Params	Sequence of Expression	M	One or more expressions evaluating to Boolean values.

Table 85: Conjunction

Disjunction is a Boolean expression that evaluates to True if an only if at least one of its parameter expressions evaluate to True. Otherwise it evaluates to False. Parameters of Disjunction are evaluated in the order they are listed and only up to the first parameter that evaluates to True. It is an error if some parameter evaluates to a value other than Boolean.

Disjunction (extends Expression)			
Property	**Type**	**M/O**	**Description**
Params	Sequence of Expression	M	One or more expressions evaluating to Boolean values.

Table 86: Disjunction

8.6.5 Work Request State Access

ActivityStateExpression is a Boolean expression which evaluates to True if an only if the specified Activity is in the specified State. It is used to specify dependencies on an Activity being in a specific State (or set of States by combining several expressions using Disjunction).

ActivityStateExpression (extends Expression)			
Property	**Type**	**M/O**	**Description**
ActivityId	Identifier	M	Identifier of the Activity being examined
StateId	Identifier	M	Identifier of the State within the Activity.

Table 87: Activity State Expression

8.7 Error Codes

FFMII interface specification defines the following error codes for inclusion in responses generated by Message Handling subsystem:

Code ID	Code Title	Meaning
Common Errors:		
E0000	OK	No Error
E0001	PARTIAL_ERROR	Some or all requests in batch operation have failed (detailed error codes are included with responses per each failing element)
E0002	INTERNAL_ERROR	Uncategorized internal error

Code ID	Code Title	Meaning
E1001	INVALID_OPERATION	Operation to be invoked not recognized by the Implementation or not available to given Manager
E1002	UNSUPPORTED_OPERATION	Operation is not supported by recipient
E1003	INVALID_DATA	Data on input is not valid or incomplete
E1004	AUTHENTICATION_FAILED	Authentication failed or authentication required / missing authentication data
Reference Data Management:		
E2001	INVALID_REPOSITORY	Unrecognized repository or invalid repository name
E2002	INVALID_OPERATION	The operation requested cannot be performed within the context of given repository or item
E2003	REPOSITORY_FULL	Unable to add more items to the repository
E2004	REPOSITORY_TABLE_FULL	Unable to add more repositories
E2005	REPOSITORY_EXISTS	Attempt to create repository that already exists in the system
E2006	REPOSITORY_NOT_EMPTY	Unable to remove requested repository due to not being empty
E2010	ACCESS_DENIED	Repository or item cannot be accessed through the Interface (even if reference to it may exist and can be used in other places)
E2011	REPOSITORY_READ_ONLY	Trying to update WriteProtected repository
E2012	INVALID_REFERENCE	Reference is not valid
Work Request Management:		
E3001	UNKNOWN_WTS	Work Request refers to an unknown Work Type Specification
E3002	UNKNOWN_WR	Operation arguments refer to an unknown Work Request
E3003	UNKNOWN_ACTIVITY	Operation arguments refer to an unknown Activity
E3004	UNKNOWN_STATE_MODEL	Activity Specification refers to an unknown State Model
E3005	UNKNOWN_ACTION	Work Type Specification or operation arguments refer to an unknown Action within the associated Work Type Specification
E3006	UNKNOWN STATE	Requests refer to an unknown State
E3007	UNKNOWN_STEP	Action Specification refers to an unknown Step within the associated Work Type Specification
E3008	UNKNOWN_DATA_ELEMENT	Work Type Specification or Work Request refers

Code ID	Code Title	Meaning
		to an unknown Data Element within the associated Work Type Specification
E3009	DUPLICATE_ACTIVITY	Work Type Specification contains more than one Activity Specification with the same identifier
E3010	DUPLICATE_STATE_MODEL	Work Type Specification contains more than one State Model Specification with the same identifier
E3011	DUPLICATE_ACTION	Work Type Specification contains more than one Action Specification with the same identifier
E3012	DUPLICATE_STATE	Work Type Specification contains more than one State Specification with the same identifier
E3013	DUPLICATE_STEP	Work Type Specification contains more than one Step Specification with the same identifier
E3014	DUPLICATE_DATA_ELEMENT	Work Type Specification contains more than one Data Element Specification with the same identifier
E3015	ILLEGAL_DATA_TYPE	Work Request data does not match the data types of the Data Elements specified by the associated Work Type Specification
E3016	VALIDATION_ERROR	Work Request data does not match the validation rules of the Data Elements specified by the associated Work Type Specification
E3017	ILLEGAL_WTS_UPDATE	Work Request update refers to or contains a Work Type Specification that is not identical to the WTS originally associated with the WR
E3018	WR_EXISTS	Work Request already exists while it was specified to WR_PUT that a new non-existing Work Request was being created.
E3019	WR_UPDATE_COLLISION	Base revision number was specified for a Work Request update but it did not match the current revision number of the associated Work Request Status Record. That is, the Work Request has been updated after the client has last read the status record.
E3020	UPDATEABLE_ELEMENTS_IN_HEADER	Elements declared as updateable have been detected in Work Request Header form
E3021	ILLEGAL_ACTION	The Action invoked using WR_INVOKE_ACTION is not available in the current Step of the identified Activity or is not currently enabled
Field-Initiated Requests Management:		
E4001	UNKNOWN_FIR_ID	Specified Field-Initiated Request identifier does

Code ID	Code Title	Meaning
		not refer to a currently active Field-Initiated Request

Table 88: Error Codes

8.8 Common Request and Response Format

This section describes the properties common to all request and response messages.

8.8.1 Requests

When invoking operations defined by the Interface, the request arguments are operation-specific. Thus, there are no common request arguments in the abstract BaseRequest class described in Table 89. Derived operation-specific request types MAY specify additional operation-specific request data.

BaseRequest (abstract base class)			
Property	**Type**	**M/O**	**Description**
			No common request properties

Table 89: BaseRequest

8.8.2 Responses

Every operation defined by the Interface returns an error code and an optional cause description in addition to operation-specific response data. The abstract BaseResponse class described in Table 90 defines the response data common to all operations. Derived operation-specific response types MAY specify additional response data.

BaseResponse (abstract base class)			
Property	**Type**	**M/O**	**Description**
ErrorCode	ErrorCode	M	Operation result as an error code. Common error codes are specified in Section 8.7.
Cause	String	O	Optional description of the error or supplementary information for an E0000 (OK) response

Table 90: BaseResponse

8.8.3 Batch Operations

A batch operation is a single type of an operation performed on one or more identified items within a single invocation. A batch operation MAY return a distinct error code for each processed item. The semantics for such operations is as if the operation was performed on each item individually. Item-specific failures MUST NOT stop the operation from being attempted on other specified items.

The following operations are batch operations.

- WR_PUT
- WR_GET_STATUS
- WR_INVOKE_ACTION
- WR_NOTIFY_STATUS
- RD_INIT
- RD_PUT
- RD_GET
- FIR_GET_RESPONSE
- FIR_NOTIFY_RESPONSE

There are also other operations updating or returning several items within a single invocation but this section does not apply to them.

For batch operations, the value of the ErrorCode property in the BaseResponse class MUST adhere to the following rules.

- E0000 (OK) MUST be returned if and only if all specified items were processed successfully.
- E0001 (PARTIAL_ERROR) MUST be returned if some or all per-item operations failed due to item-specific error.
- Error code other than E0000 and E0001 (see Section 8.7 for defined error codes) MUST be returned if the batch operation fails on other than item-specific error.

Additionally, operation-specific response classes of batch operations contain an optional “Results” property, as shown in Table 91.

BatchOperationResponse (template, extends BaseResponse)			
Property	**Type**	**M/O**	**Description**
Results	Sequence of operation-specific result classes extending BatchItemResult	O	Per-item results of this batch operation. The structure of results depends on operation. See the result data types defined for operations domains.

Table 91: BatchOperationResponse

If a BatchOperationResponse contains an error code E0000 or E0001 in ErrorCode property of the BaseResponse class then its Results property MUST contain an operation-specific result, derived from the abstract BatchItemResult described in Table 92, for each item specified in the corresponding batch

operation request. If an error code other than E0000 or E0001 is returned in ErrorCode property of the BaseResponse class then the Results property SHOULD NOT be included in the response.

BatchItemResult (abstract base class)			
Property	**Type**	**M/O**	**Description**
Id	Identifier	M	Identifier of the item being accessed
ErrorCode	ErrorCode	M	Result of per-item operation. Error codes are specified in Section 8.7.
Cause	String	O	Optional description of the error or supplementary information for an E0000 (OK) response

Table 92: BatchItemResult

8.9 "Users" Repository

8.9.1 Introduction

Repository with identifier "`Users`" contains configuration data necessary to authenticate, authorize and communicate with individual Assignees and administrative users. Type of Values of Reference Data Items stored in the repository MUST be UserProfile. Each profile MUST have a unique identifier, which can consequently be included in Work Requests to indicate work ownership. UserProfiles also contain information required for authentication and authorization purposes. Other data suitable for this registry is description of the type of terminals the Assignee can use, and general or device-specific presets and user preferences.

8.9.2 Repository Descriptor

User repository MUST use the following standard Repository Descriptor:

`Id`	`"Users"`
`Description`	`"Catalogue of user profiles"`
`Readable`	`<implementation specific>`
`Writable`	`<implementation specific>`
`CustomProperties`	`<implementation specific>`

Table 93: Users Repository

8.9.3 Visibility and access rules

The Implementation MUST support "Users" repository internally.

The Implementation MAY support RDM for publishing user profiles. Readable flag indicates whether user profiles are readable over RDM. Writable flag indicates whether user profiles are updateable over RDM. If user profiles are not published over RDM then it is assumed that user information is synchronized in an implementation-specific manner.

An Implementation MAY restrict access to user profiles repository on per-Manager basis.

8.9.4 Data types

This section describes data types defined in the context of user profiles repository.

UserProfile			
Property	**Type**	**M/O**	**Description**
PersonalInfo	PersonalInfoRec	M	Basic identification of the user
UserRoles	Sequence of Identifier	M	Roles granted to the user. The roles define the privileges the user has as well as determine the views and functions that are relevant for the user. The order in which the roles have been listed is not significant. See section 8.9.5 for available roles
DeviceProfiles	Sequence of DeviceProfile	M	Describes what types of devices (and client) technologies can be used for communication with the Assignee
Credentials	Sequence of CredentialSpec	M	All credentials (or authentication services references) that can be used for authenticating the user NOTE: Some credential types may be applicable for certain types of clients only (e.g. WebUI, mobile client, etc…)
Preferences	UsagePreferences	O	General usage preferences. These may be overridden by device specific usage preferences in Device Profiles. If not specified, auto-detected usage preferences or implementation-specific defaults are used. (see note below for additional details)
EnabledSince	DateTime	O	The user is enabled and active after this date and time. Before that the user record should be treated as if it did not exist If not specified, the user is active immediately
EnabledUntil	DateTime	O	The user is enabled and active until this date and time. After that the user record should be treated as expired If not specified, the user is valid indefinitely (has no expiration point set)
CustomProperties	Dictionary	O	Implementation-specific custom properties

Table 94: UserProfile

Note: Usage Preferences describe general or device specific user preferences for the Implementation. The Implementation SHOULD take into account usage preferences specified in the user profile, and customize the usage experience accordingly.

If a specified preference setting is not supported then the Implementation MUST fail gracefully by defaulting to a known working setting.

1741 If some of the preferences have not been specified then the Implementation SHOULD use the
1742 information provided by the user client to detect user preferences, if possible.

PersonalInfoRec			
Property	**Type**	**M/O**	**Description**
FirstName	String	M	First name of the user
MiddleName	String	O	Middle name(s) or middle initial(s) of the user
LastName	String	M	Last name of the user
ShortName	String	O	If provided, used in place of the full name

1743 **Table 95: PersonalInfoRec**

DeviceProfile			
Property	**Type**	**M/O**	**Description**
Id	Identifier	M	Unique identifier for this device profile
Enabled	Boolean	O	Whether this device profile is enabled. If not specified, default is True.
DeviceTypeId	Identifier	O	Implementation-specific device type identifier
Preferences	UsagePreferences	O	Device specific usage preferences that override general Preferences specified in the User Profile by individual preference Property. E.g. if PreferredLocales and Theme are specified in the User Profile, and the Device Specific Preferences only specify Theme, PreferredLocales in the User Profile locale will be applied.. If not specified, any UsagePreferences are used instead. In the absence of user-level and general preferences, any automatically determined preferences, or implementation-specific defaults are used
CustomProperties	Dictionary	O	Implementation-specific custom properties

1744 **Table 96: DeviceProfile**

CredentialSpec (abstract base class)			
Property	**Type**	**M/O**	**Description**
Id	Identifier	M	Unique identifier within the context of the UserProfile
Enabled	Boolean	M	Administratively enabled for use
ValidUntil	DateTime	O	The date and time after which this credential is not valid (if not specified, no time limit exists for the validity of this credential)
Locked	Boolean	M	If set to “True”, the credential is locked due to breach of policies (e.g. number of failed login attempts exceeded) Defaults to “False” when a new entry is created, or when no policy framework is provided by the Implementation

1745 **Table 97: CredentialSpec**

PasswordCredentialSpec (extends CredentialSpec)			
Property	**Type**	**M/O**	**Description**
LoginName	String	M	User Login Name
Password	String	O	User Password An Implementation SHOULD NOT expose existing password information, unless otherwise configured using an Implementation specific mechanism. If a password is not to be exposed or if the password to be set is disabled, this property MUST be omitted.
PasswordProtectionScheme	Identifier	O	Indicates whether the password is delivered by the Manager as plain text (default) or secure hash of particular type Predefined values are: - “Plain” (default) - “MD5” - “SHA” Implementation-specific identifiers MAY be used and they MUST start with prefix “X-“

1746 **Table 98: PasswordCredentialSpec**

UsagePreferences			
Property	**Type**	**M/O**	**Description**
PreferredLocales	Sequence of LocaleSpecifier	O	Sequence of the preferred locales to be used for the user interface and the content, in descending order of preference.
Theme	Identifier	O	Chooses one of the different visual themes that are made available by the Implementation. Themes can be used either to create a familiar user experience (such as a company-specific theme) or to choose a special visual experience tailored for a group of users (such as a high contrast theme). Available themes are implementation-specific

1747 **Table 99: UsagePreferences**

1748 8.9.5 User Roles

1749 The following generic user roles are predefined. The Implementation MAY support the predefined roles.
1750 Implementation-specific roles MAY be introduced and they MUST have identifiers starting with prefix “X-“.

1751

Role	Description	Notes
Assignee	Assignee can receive Work Requests and process them according to the Work Type Specification, and initiate Field-Initiated Requests.	
TeamLeader	TeamLeader may have additional privileges for accessing Work Requests assigned to other Assignees	
WorkPlanner	WorkPlanner is responsible for planning and scheduling execution of work	
UserAdmin	UserAdmin can administer and maintain the user registry, if Implementation provides a local user interface for this	
SystemAdmin	SystemAdmin can administer and maintain system parameters and Reference Data repositories, if Implementation provides a local user interface for this	

1752 **Table 100: User Roles**

1753 8.9.6 Repository-specific semantics and restrictions

1754 Only UserProfile type of values can be stored in this repository.

1755 The semantics of references within this repository are out of scope of this specification.

1756 8.10 “WorkTypes” repository

1757 8.10.1 Introduction

1758 A Reference Data repository with identifier ”WorkTypes”, also known as the Work Type Repository,
1759 contains information about registered Work Types that may be shared among multiple Work Requests.
1760 Type of Values of Reference Data Items stored to the repository MUST be WorkTypeSpecification. A Work

1761 Request MAY contain a reference to a Work Type specified in this repository.

1762 8.10.2 Repository descriptor

1763 The Work Type Repository MUST use the following standard Repository Descriptor:

Id	"WorkTypes"
Description	"Catalogue of work types"
Readable	<implementation specific>
Writable	<implementation specific>
CustomProperties	<implementation specific>

1764 **Table 101: WorkTypes Repository**

1765 8.10.3 Visibility and access rules

1766 The Implementation MAY support the Work Type Repository.

1767 Work Types stored into the Work Type Repository by a specific Manager MUST be available to that
1768 Manager. However, the Implementation SHOULD, by default, hide these entries from other Managers
1769 and allow different Managers to use overlapping Work Type identifiers. In this case any Work Type
1770 reference MUST be interpreted in the context of the requesting Manager.

1771 The Implementation MAY also support alternative configurable implementation-specific visibility policies.
1772 A Manager using the Work Type Repository for Manager-specific Work Types MUST be able to generate
1773 identifiers containing an element that uniquely identifies the Manager. For example, to support a
1774 configurable prefix for the identifiers.

1775 The Implementation MAY impose Manager-specific access restrictions for the Work Type Repository.
1776 Readable flag indicates whether the requesting Manager can read Work Type Repository entries over
1777 RDM. Writable flag indicates whether the requesting Manager is allowed to update Work Type Repository
1778 entries over RDM.

1779 The Implementation MAY also provide implementation-specific means for statically initializing and
1780 modifying the contents of the Work Type Repository in addition to RDM based interface.

1781 8.10.4 Data types

1782 See definition of WorkTypeSpecification in Section 8.2.1.1.

1783 8.10.5 Repository-specific semantics and restrictions

1784 Only values of type WorkTypeSpecification can be stored in this repository.

1785 The semantics of references within this repository are out of scope of this specification.

1786 8.11 "FieldInitiatedRequests" Repository

1787 8.11.1 Introduction

1788 A Reference Data repository with identifier "FieldInitiatedRequests", also known as the Field-Initiated
1789 Requests Repository, contains specifications of registered activities that the Implementation may request
1790 a Manager to perform if the Manager supports Field-Initiated Requests. Type of Values of Reference Data
1791 Items stored in this repository MUST be Field-Initiated Request Specification. A Work Request MAY
1792 contain a reference to a Field-Initiated Request Specification specified in this repository.

1793 The Implementation MAY support Field-Initiated Requests. If so, it MUST support a Field-Initiated

1794 Requests repository.

1795 8.11.2 Repository Descriptor

1796 The Field-Initiated Requests Repository MUST use the following standard Repository Descriptor:

Id	"FieldInitiatedRequests"
Description	"Catalogue of Field-Initiated Requests"
Readable	<implementation specific>
Writable	<implementation specific>
CustomProperties	<implementation specific>

1797 **Table 102: Field-Initiated Requests Repository**

1798 8.11.3 Visibility and access rules

1799 The Implementation MAY support the Field-Initiated Requests Repository.

1800 Field-Initiated Request Specifications stored into the Field-Initiated Requests Repository by a specific
1801 Manager MUST be available to that Manager. However, the Implementation SHOULD, by default, hide
1802 these entries from other Managers and allow different Managers to use overlapping Field-Initiated
1803 Request identifiers. In this case any reference to Field-Initiated Request MUST be interpreted in the
1804 context of the requesting Manager.

1805 The Implementation MAY also support alternative configurable implementation-specific visibility policies.
1806 A Manager using the Field-Initiated Requests Repository for Manager-specific Field-Initiated Request
1807 entries MUST be able to generate identifiers containing an element that uniquely identifies the Manager.
1808 For example, to support a configurable prefix for the identifiers.

1809 The Implementation MAY impose Manager-specific access restrictions for the Field-Initiated Requests
1810 Repository. Readable flag indicates whether the requesting Manager can read Field-Initiated Request
1811 entries over RDM. Writable flag indicates whether the requesting Manager is allowed to update Field-
1812 Initiated Request entries over RDM.

1813 The Implementation MAY also provide implementation-specific means for statically initializing the Field-
1814 Initiated Requests in addition to RDM based interface.

1815 8.11.4 Data types

1816 See definition of FieldInitiatedRequestSpecification data type in Section 8.4.1.1.

1817 8.11.5 Repository-specific semantics and restrictions

1818 Only values of type FieldInitiatedRequestSpecification can be stored in this repository.

1819 The semantics of references within this repository are out of scope of this specification.

1820 An Implementation MUST NOT send a Field-Initiated Request to a Manager that does not support that
1821 type of Field-Initiated Request, i.e. based on Field-Initiated Request Specification not registered by the
1822 Manager.

9 FFMII Protocol Bindings

Protocol Binding and Payload Encapsulation layer performs conversion of command sequences, input parameters, error codes and return values between their wire line and software-level representations.

This version of the Interface specification supports a single protocol binding, SOAP, as listed in Table 103.

Protocol	Requirement	Normative Reference
SOAP (Web Services)	mandatory	[SOAP]

Table 103: Protocol Binding and Payload Encapsulation

9.1 SOAP over HTTP (Web Service)

Table 104 summarizes transport protocols recognized by FFMII specification and their required support level:

Protocol	Requirement	Normative Reference
HTTPS	mandatory	[RFC2818]
HTTP	optional	[RFC2616]

Table 104: Transport Protocol

The method of discovering end-points is out of the scope of the FFMII specification.

Other transport protocols MAY be supported.

9.1.1 XML namespace

All the XML types and services use fully-qualified XML names. The following XML namespaces are used in the SOAP binding.

Namespace	Description
http://docs.oasis-open.org/ffm/ns/v1.0/ws/implementation	FFMII web service and port declarations for the Implementation
http://docs.oasis-open.org/ffm/ns/v1.0/ws/manager	FFMII web service and port declarations for the Manager
http://docs.oasis-open.org/ffm/ns/v1.0/common/api	Operation invocation related XML types shared by different parts of the Interface. Includes abstract base types for request objects, response objects and client credentials.
http://docs.oasis-open.org/ffm/ns/v1.0/common/model	Domain model XML types shared by different parts of the Interface. Includes core data types such as Dictionary and MultiLanguageString.
http://docs.oasis-open.org/ffm/ns/v1.0/system/info/api	Operation invocation related XML and elements types for system information operations. Includes related request and response objects.

Namespace	Description
http://docs.oasis-open.org/ffm/ns/v1.0/system/info/model	Domain model XML types for system information operations. Includes IdentityDescriptor.
http://docs.oasis-open.org/ffm/ns/v1.0/system/capability/api	Operation invocation related XML types and elements for capability management. Includes related request and response objects.
http://docs.oasis-open.org/ffm/ns/v1.0/system/capability/model	Domain model XML types for capability management. Includes CapabilityDescriptor.
http://docs.oasis-open.org/ffm/ns/v1.0/wrm/api	Operation invocation related XML types and elements for Work Request Management. Includes related request and response objects.
http://docs.oasis-open.org/ffm/ns/v1.0/wrm/model	Domain model XML types for Work Request Management. Includes Work Type and Work Request related objects.
http://docs.oasis-open.org/ffm/ns/v1.0/firm/api	Operation invocation related XML types and elements for Field-Initiated Request. Includes related request and response objects.
http://docs.oasis-open.org/ffm/ns/v1.0/firm/model	Domain model XML types for Field-Initiated request. Includes Field-Initiated Requests domain objects.
http://docs.oasis-open.org/ffm/ns/v1.0/rdm/api	Operation invocation related XML types and elements for Reference Data Management. Includes related request and response objects.
http://docs.oasis-open.org/ffm/ns/v1.0/rdm/model	Domain model XML types for Reference Data Management. Includes Reference Data related objects.
http://docs.oasis-open.org/ffm/ns/v1.0/rdm/model/profile	User profile related XML types.

Table 105: XML namespaces

9.1.2 Parameter Encoding

SOAP requests use document literal encoding for request and response parameters. Each Interface operation has distinct request and response types that utilize shared abstract base types as applicable.

9.1.3 Data types and operations

The primitive data types are mapped to the corresponding XML Schema types. Other data types are declared in WSDL as complex XML Schema types.

The operations supported by Implementation are declared in WSDL as operations bound to a single service called “FieldForceManagementImplementationService”.

The operations supported by Manager are declared in WSDL as operations bound to a single service called “FieldForceManagementManagerService”.

9.2 Primitive and derived data types

The primitive data types specified in section 7.2 are mapped directly to the corresponding XML Schema types [Schema2] according to the following table.

Primitive Types	XML Schema Type
String	xs:string
Integer	xs:int
Double	xs:double
Decimal	xs:decimal
Boolean	xs:boolean
DateTime	xs:dateTime
Date	xs:date
Time	xs:time
Duration	xs:duration
Binary	xs:base64Binary

Table 106: Primitive Data types / XML schema types mappings

Derived data types specified in Section 7.3 are mapped to XML types that extend the corresponding XML base type. Additional XML types have been specified to model properties of complex types that only accept an enumerated set of valid values of type Identifier.

1856 Table 107 lists derived data types and the corresponding XML types.

Derived Type	XML Schema Type	Notes
Identifier	Identifier	The XML schema does not enforce Identifier well-formedness. The Manager or Implementation sending or receiving data is responsible for enforcing well-formedness. Additional derived XML types have been specified for some enumerated value sets. These types are listed below.
	BinaryOperator	BinaryOperatorExpression property Operator
	DataChangeHistoryEntryCause	DataChangeHistoryEntry property Cause
	DataFieldType	DataFieldSpecification property Type
	FieldInitiatedRequestSpecificationType	FieldInitiatedRequestSpecification property FIRType
	GenericActionType	ActionSpecification property GenericType
	InsertPosition	AddRowOperation property InsertPosition
	StatusCategory	StateSpecification property StatusCategory
	SystemType	IdentityDescriptor property SystemType
	UnaryOperator	UnaryOperatorExpression property Operator
ErrorCode	ErrorCode	
LocaleSpecifier	LocaleSpecifier	

1857 **Table 107: Derived data types / XML schema types mappings**

1858 9.3 Composite and specialized data types

1859 The composite types specified in Section 7.4 are mapped to corresponding XML structures or complex
1860 types according to the following mapping.

Composite Types	XML Schema Type	Notes
Class	Classes are modeled as complex XML types with the same local name as used for the class. Properties are modeled as contained elements. Properties accepting more than one type of a value derived from a common base type “Type” use a container XML type “AnyType”. The container type includes a choice of elements each having the name and type of one compatible type. For example, properties accepting any primitive value use the type	See Section 9.1.1 for used namespaces

Composite Types	XML Schema Type	Notes
	AnyPrimitive that accepts content such as “<Int>3</Int>” or “<String>Example</String>”.	
Dictionary	Type Dictionary in namespace http://docs.oasis-open.org/ffm/ns/v1.0/common/model	
Sequence	Sequences of some particular type “Type” are modeled as an XML type “TypeSequence” containing a sequence of elements having the name and type of the specified contained type. Sequences accepting more than one type of values are specified as choice of elements each having the name and type of one compatible type.	See Section 9.1.1 for used namespaces

Table 108: Composite Type / XML Structure Mappings

Specialized data types specified in Section 7.5 are modeled like a corresponding class with the exception of the MLText data type.

The MLText data type specified in Section 7.5.1 uses a specialized XML binding to produce simplified XML. The XML type MLText does not have a container element for the Values Sequence. Instead the associated values are directly contained in MLText as elements having the name Value and type LocalizedString. The XML type LocalizedString extends the simple XML Schema type xs:string directly, containing the Value and having the Locale property as an attribute.

9.4 Operations

The operations have been mapped to the following web service operations to unify the naming convention used for types and operations.

FFMII Operation	Web Service Operation	Request/Response namespace
SYS_INFO_GET	SysInfoGet	http://docs.oasis-open.org/ffm/ns/v1.0/system/info/api
SYS_CAPA_GET	SysCapaGet	http://docs.oasis-open.org/ffm/ns/v1.0/system/capability/api
WR_PUT	WrPut	http://docs.oasis-open.org/ffm/ns/v1.0/wrm/api
WR_LIST	WrList	http://docs.oasis-open.org/ffm/ns/v1.0/wrm/api
WR_GET_STATUS	WrGetStatus	http://docs.oasis-open.org/ffm/ns/v1.0/wrm/api
WR_INVOKE_ACTION	WrInvokeAction	http://docs.oasis-open.org/ffm/ns/v1.0/wrm/api
WR_NOTIFY_STATUS	WrNofityStatus	http://docs.oasis-open.org/ffm/ns/v1.0/wrm/api
RD_INIT	RdInit	http://docs.oasis-

FFMII Operation	Web Service Operation	Request/Response namespace
		open.org/ffm/ns/v1.0/rdm/api
RD_PUT	RdPut	http://docs.oasis-open.org/ffm/ns/v1.0/rdm/api
RD_DELETE	RdDelete	http://docs.oasis-open.org/ffm/ns/v1.0/rdm/api
RD_LIST	RdList	http://docs.oasis-open.org/ffm/ns/v1.0/rdm/api
RD_GET	RdGet	http://docs.oasis-open.org/ffm/ns/v1.0/rdm/api
RD_REFS_GET	RdRefsGet	http://docs.oasis-open.org/ffm/ns/v1.0/rdm/api
RD_REFS_PUT	RdRefsPut	http://docs.oasis-open.org/ffm/ns/v1.0/rdm/api
RD_REFS_DELETE	RdRefsDelete	http://docs.oasis-open.org/ffm/ns/v1.0/rdm/api
RD_REPO_LIST	RdRepoList	http://docs.oasis-open.org/ffm/ns/v1.0/rdm/api
RD_REPO_CREATE	RdRepoCreate	http://docs.oasis-open.org/ffm/ns/v1.0/rdm/api
RD_REPO_DELETE	RdRepoDelete	http://docs.oasis-open.org/ffm/ns/v1.0/rdm/api
RD_REPO_UPDATE_PROPS	RdRepoUpdateProps	http://docs.oasis-open.org/ffm/ns/v1.0/rdm/api
FIR_EXECUTE	FirExecute	http://docs.oasis-open.org/ffm/ns/v1.0/firm/api
FIR_GET_RESPONSE	FirGetResponse	http://docs.oasis-open.org/ffm/ns/v1.0/firm/api
FIR_NOTIFY_RESPONSE	FirNotifyResponse	http://docs.oasis-open.org/ffm/ns/v1.0/firm/api

1872 **Table 109: Operations / Web Services Mappings**

1873 For each operation, there is a corresponding request and response XML type named after the operation.
1874 For example, for “SysCapaGet” operation there is a request XML type “SysCapaGetRequest” and a
1875 response XML type “SysCapaGetResponse” in the “http://docs.oasis-open.org/ffm/ns/v1.0/capability/api”
1876 namespace. The request type defines the operation arguments and the response type the response data.
1877 All request and response types are based on common abstract types “BaseRequest” and
1878 “BaseResponse” in the “http://docs.oasis-open.org/ffm/ns/v1.0/common/api” namespace. Additional
1879 intermediate abstract types have been used where applicable.

1880 9.5 Authentication

1881 The Manager or Implementation making a web service request MAY supply a X.509 client certificate
1882 which is transmitted and verified as part of the two-way TLS/SSL certificate exchange.

1883 The Manager or Implementation making a web service request MAY supply a username and a plaintext
1884 password in SOAP headers conforming to the Username Token Profile [WSS-UTP] of OASIS Web
1885 Services Security [WSS]. This method SHOULD only be used with TLS/SSL secured or otherwise secure
1886 transport layer.

1887 The Manager or Implementation making a web service request MAY authenticate using other security
1888 profiles defined in OASIS Web Services Security [WSS].

1889 The Implementation or Manager capable of receiving web service requests SHOULD support two-way
1890 TLS/SSL certificate exchange and username based authentication conforming to the Username Token
1891 Profile [WSS-UTP] with plaintext password and MAY support other security profiles defined in OASIS
1892 Web Services Security [WSS].

1893 The Manager or Implementation making a web service request SHOULD use TLS/SSL and check the
1894 received server certificate against the configured server identity or trust chain before sending request
1895 data to the server in order to authenticate the server and to prevent sensitive information from being
1896 exposed.

1897 9.6 WSDL Files

1898 The WSDL description of the FFMII interface is included in an archive [FFMII-WSDL] containing the
1899 WSDL for both Implementation and Manager interfaces as well as the XML Schema type definitions for
1900 the associated XML namespaces.

1901 Notice that some operations declared in the interfaces are optional and a Manager does not necessarily
1902 have a web service endpoint at all.

1903 The following files contain the WSDL description of the Implementation and Manager interfaces.

WSDL File	Description
FFMII_v1_0_ws_implementation.wsdl	Web service interface of the Implementation
FFMII_v1_0_ws_manager.wsdl	Web service interface of the Manager

1904 **Table 110: WSDL Files**

1905

10 Conformance

An implementation of Manager or Implementation conforms to this specification if it meets all of the following requirements:

1) It meets the requirements indicated throughout this specification by keywords specified in Section 3.1.1, and
2) it complies with semantics and structure of operations and data types specified in Sections 5, 6, 7 and 8, and
3) it is compatible with the web service binding specified in Section 9.

Appendix A. Acknowledgements

The following individuals have participated in the creation of this specification and are gratefully acknowledged:

Participants:

Liat Zahavi-Barzily	ClickSoftware Technologies Ltd
Israel Beniaminy	ClickSoftware Technologies Ltd.
Jiri Hlusi,	Nokia Siemens Networks GmbH & Co. KG
Johannes Lehtinen	Rossum Oy
Thinh Nguyenphu	Nokia Siemens Networks GmbH & Co. KG
Ilkka Salminen	Newelo Oy
Jose Siles	Nokia Siemens Networks GmbH & Co. KG
Juha Tiihonen	Aalto University Foundation
Sami Vaskuu	Newelo Oy

Appendix B. Revision History

Revision	Date	Editor	Changes Made
01	26 August 2011	Thinh Nguyenphu	Initial draft based on FFMIISpecification-Foundation20110720-D
01	11 September, 2011	Thinh Nguyenphu	Input from FFMII-Specification-Foundation-20110826-v1.0-wd01--architecture-upd2 FFMII-Specification-Foundation-20110826-v1.0-wd01--architecture-upd2
01	30 September, 2011	Thinh Nguyenphu	Input from FFMII-Specification-Foundation-20110919-JL-statushandling02 FFMII-Specification-Foundation-20110826-v1.0-wd01--CoreDataModel-upd2
01	06 October, 2011	Thinh Nguyenphu	Input from FFMII-Specification-Foundation-20110930-v1.0-wd01-IB-20111004 FFMII-Specification-Foundation-20110930-v1.0-wd01-FieldRequest-rev02
01	12 October, 2011	Thinh Nguyenphu	Input from FFMII-Specification-Foundation-20110930-v1.0-wd01-statemodel-edits-01 FFMII-Specification-Foundation-20111006-v1.0-wd01_work-req-status-change-notification-edits-02 Editorial clean up (fixing fonts, references, etc.)
01	26 October, 2011	Thinh Nguyenphu	Input from FFMII-Specification-Foundation-20111012-v1.0-wd01 Capablities-01_JTComments-02 FFMII-Specification-Foundation-20111017-v1 0-wd01 Error Codes -02 FFMII-Specification-Foundation-20111017-v1.0-wd01 Protocol Binding-02 FFMII-Specification-Foundation-20111017-v1.0-wd01Data Forms-02 FFMII-Specification-Foundation-20111012-v1.0-wd01 WRM-02 Editorial fix all of figures
01	01 November,	Thinh Nguyenphu	FFMII-Specification-Foundation-20111012-v1.0-wd01-RequestResponse-02

	2011		Diagram of States, Steps and Actions v02 FFMII Status indicator explanations IB-v04 FFMII-Specification-Foundation-20111017-v1.0-wd01Data Forms-04 FFMII-Specification-Foundation-20110911-v1.0-wd01-IB-13Sep11 FFMII-Specification-Foundation-20111012-v1.0-wd01-RDM-02 FFMII-Specification-Foundation-20111012-v1.0-wd01-WRM-04-StatusRecord
01	17 November 2011	Thinh Nguyenphu	http://lists.oasis-open.org/archives/ffm/201111/msg00011.html http://lists.oasis-open.org/archives/ffm/201111/msg00016.html FFMII-Specification-Foundation-20111101-v1.0-wd01-AvailableTopics-01.docx
01	29 November, 2011	Thinh Nguyenphu	FFMII-Specification-Foundation-20111117-v1.0-wd01 editcleanup sec6_4_3_1-01 FFMII-Specification-Foundation-20111117-v1.0-wd01 Namespace-01 FFMII-Specification-Foundation-20111101-v1.0-wd01-IB-workorder+supplementary+locationtype
01	13 December, 2011	Thinh Nguyenphu	FFMII-Specification-Foundation-20111129-v1.0-wd01_multichoicealternative-edits-02 FFMII-Specification-Foundation-20111129-v1.0-wd01-WorkTypeRepository-01 FFMII-Specification-Foundation-20111129-v1.0-wd01-ActionAvailable-02 FFMII-Specification-Expressions-03
01	19 December, 2011	Thinh Nguyenphu	FFMII-Specification-Foundation-20111129-v1.0-wd01-ChangeNotificationManagement-02 FFMII-Specification-Foundation-201111214-v2.0-wd01-ib-fir FFMII-Specification-Foundation-201111214-v1.0-wd01-ExpressionEdits-01
01	19 January 2012	Thinh Nguyenphu	FFMII-Specification-Foundation-201111214-v4.0-wd01-ib-fir FFMII-Specification-Foundation-201111219-v1.0-wd01-JL-WTRepoEdits-WSS-WSDL-02
01	27 January 2012	Thinh Nguyenphu	FFMII-Specification-Foundation-20120124-v1.0-wd01-cleanup-02-Clean FFMII-Specification-Foundation-20120119-v1.0-wd01-JL_WSDL_edits-02
01	14 March 2012	Thinh Nguyenphu	FFMII-Specification-20120127-v1.0-wd01Clean_JT_Comments_05
01	15 March 2012	Thinh Nguyenphu	https://lists.oasis-open.org/archives/ffm/201202/msg00021.html FFMII-Specification-20120306-v1.0-wd01Clean-IB-charset-

			definition FFMII-Specification-20120127-v1.0-wd01Clean ThinhComments-02: (Section 1 – 8.1.4)
01	22 March 2012	Thinh Nguyenphu	FFMII-Specification-20120127-v1.0-wd01Clean ThinhComments-03 (Section 8.2 to end)
01	29 March 2012	Thinh Nguyenphu	FFMII-Specification-20120127-v1.0-wd01Clean-JL-Review-05
01	04 April 2012	Thinh Nguyenphu	FFMII-Specification-20120329-v1 0-wd01Thinh clean-up FFMII-Specification-20120315-v1.0-wd01Clean_JuhaActions-02 FFMII-Specification-20120322-v1.0-wd01_JS FFMII-Specification-20120329-v1 0-wd01-JL-20120402-02 FFMII-Specification-20120329-v1_0-wd01-JTSequence
01	11 April 2012	Thinh Nguyenphu	https://lists.oasis-open.org/archives/ffm/201204/msg00018.html https://lists.oasis-open.org/archives/ffm/201204/msg00021.html https://lists.oasis-open.org/archives/ffm/201204/msg00024.html
01	08 May 2012	Thinh Nguyenphu	https://lists.oasis-open.org/archives/ffm/201204/msg00046.html FFMII-Specification-20120411-v1 0-wd01_JT_CHECKWSDL_NOTES-02 FFMII-Specification-20120411-v1 0-wd01 IB comments (after discussion) FFMII-Specification-20120411-v1 0-wd01 Thinh Comments v02 FFMII-Specification-20120411-v1 0-wd01-JL_WSDL_review_01 FFMII-Specification-20120411-v1 0-wd01_JL_entity_20120417_01 FFMII-Specification-20120411-v1 0-wd01_JL_input_20120418_01 FFMII-Specification-20120411-v1 0-wd01_JL_xmltypes_20120424_01 FFMII-Specification-20120411-v1 0-wd01_JL_topics_20120424_01 FFMII-Specification-20120411-v1 0-wd01_JT_task_to_informal_concept-02
02	21 May 2012	Thinh Nguyenphu	Global editorial clean-up (formats, page breaks, etc.) Conformance Section
02	23 May 2012	Thinh Nguyenphu	Conformance Section
03	29 June 2012	Thinh Nguyenphu	Fixed all of broken reference links and editorial clean up on all of figures, based on 30 days public review comments.
03	09 July 2012	Thinh Nguyenphu	FFMII-Specification-20120629-v1 0-wd03_JL_RDfix_20120707-01
04	09 September 2012	Thinh Nguyenphu	Editorial clean per TC Admin

05	12 September 2012	Thinh Nguyenphu	Editorial clean up references links and Acknowledgement list
06	20 September 2012	Thinh Nguyenphu	Fixed broken link reference at lines 1230, 1828 and 1831.

1930