

**Deliverable D3.3**

**Runtime and Hyperty Messaging Node Phase 2**

|  |  |
| --- | --- |
| Editor: | Paulo Chainho, PT Inovação |
| Deliverable nature: | <(R) Document, Report ¦ (DEM) Demonstrator ¦ (DEC) Website, press, media ¦ (OTHER) Software> |
| Dissemination level: (Confidentiality) | Public (PU) |
| Contractual delivery date: | 28/02/2017 |
| Actual delivery date: | 28/02/2017 |
| Suggested readers: | Service providers’ designers and developers |
| Version: | 0.1 |
| Total number of pages: |  |
| Keywords: | Hyperty, Microservices, Edge Computing, Runtime, Messaging Middleware, Protofly, Reporter – Observer, Data Synchronisation |

***Abstract***

This Deliverable contains documentation that accompanies Phase 2 release of reTHINK Core Framework components published as source code in reTHINK Github repositories. The reTHINK Core Framework Phase 2 release includes the Hyperty Runtime (Core Runtime Components that are reused in Hyperty Browser Runtime and Hyperty NodeJS Runtime) and three Message Node implementations: Vertx Message Node, Matrix Message Node, and NodeJS Message Node, as well as the Hyperty Service Framework featuring a comprehensive set of application program interfaces (APIs) and JavaScript libraries to facilitate the development of Hyperties. This deliverable includes a full suite of tutorials targeting external developers; a summary of major changes done in the main specification of reTHINK Core Framework components; and the documentation of released components. The reTHINK Core Framework Phase 2 components are used to implement Phase 2 reTHINK scenarios that will be used to validate reTHINK Framework with Telcos and start-ups.

[End of abstract]

Disclaimer

This document contains material, which is the copyright of certain reTHINK consortium parties, and may not be reproduced or copied without permission.

All reTHINK consortium parties have agreed to full publication of this document.

The commercial use of any information contained in this document may require a license from the proprietor of that information.

Neither the reTHINK consortium as a whole, nor a certain part of the reTHINK consortium, warrant that the information contained in this document is capable of use, nor that use of the information is free from risk, accepting no liability for loss or damage suffered by any person using this information.

*This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 645342. This publication reflects only the author’s view and the European Commission is not responsible for any use that may be made of the information it contains.*

Impressum

Full project title: **Trustful hyper-linked entities in dynamic networks**

Short project title: **reTHINK**

Number and title of work-package: **WP3 – Core Framework Implementation**

Number and title of task: **Task 3.2 Hyperty runtime prototype, Task 3.5 Hyperty Messaging Node, Task 3.4 Hyperty Services Framework**

Document title: **Runtime and Hyperty Messaging Node Phase 2**

Editor: **Paulo Chainho, PT Inovação**

Work-package leader: **Paulo Chainho**, company: **PT Inovação**

Copyright notice

2016 Participants in project RETHINK

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License. To view a copy of this license, visit [http://creativecommons.org/licenses/by-nc-nd/3.0](http://creativecommons.org/licenses/by-nc-nd/3.0/)

# Executive summary

This Deliverable accompanies Phase 2 release of the Core Framework components published in reTHINK Github repositories, including the Hyperty Runtime (Core Runtime Components that are reused in Hyperty Browser Runtime and Hyperty NodeJS Runtime) and three Message Node implementations: Vertx Message Node, Matrix Message Node, and NodeJS Message Node. The Hyperty Service Framework is also released and it features a comprehensive set of application program interfaces (APIs) and JavaScript libraries to facilitate the development of Hyperties. A full suite of documentation targeting external developers is included in this report. Besides, this document provides an overview of changes done in the main specification of reTHINK Core Framework components according to feedback taken from the implementation activity.

The reThink source code repositories associated with this deliverable are publicly available on GitHub:

* Hyperty Core Runtime: <https://github.com/reTHINK-project/dev-runtime-core>
* Hyperty Runtime Browser: <https://github.com/reTHINK-project/dev-runtime-browser>
* Hyperty Runtime NodeJS: <https://github.com/reTHINK-project/dev-runtime-nodejs>
* Vertx Message Node: <https://github.com/reTHINK-project/dev-msg-node-vertx>
* Matrix Message Node : <https://github.com/reTHINK-project/dev-msg-node-matrix>
* NodeJS Message Node: <https://github.com/reTHINK-project/dev-msg-node-nodejs>
* Hyperty Service Framework: <https://github.com/reTHINK-project/dev-service-framework>

All code is provided under the Apache 2 license to assure most flexible use in commercial environments that may build upon reTHINK components without being obliged to publish modifications and enhancements.

# List of authors

|  |  |  |
| --- | --- | --- |
| Company | Author | Contribution |
| PTIN | Paulo Chainho  Miguel Mesquita | Syncher, Runtime Loader, Message Bus, MiniBus, RuntimeUA, Sync Manager, Vertx Messaging Node, Tutorials |
| Fraunhofer | Alice Cheambe  Robert Ende  Mark Emmelmann  Paul Zühlcke | Message Factory, Catalogue Data Object Factory, Runtime Catalogue, Tutorials |
| Quobis | Anton Roman Portabales  Andres Estevez  David Vílchez | Address Factory, Browser Runtime, Standalone Runtime, Persistence Manager, Tutorials |
| Apizee | Arnaud Vallée  Nicolas Daviaud  Frédéric Luart | NodeJS Messaging Node |
| Deutsche Telekom | Steffen Druesedow  Philipp Dockhor  Johannes Hamfler  Kay Haensge | Matrix Messaging Node, Tutorials |
| Orange Labs | Simon Bécot  Kevin Corre | Identity Module, IdP Proxy, Tutorials |
| INESC-ID | Ana Caldeira  Ricardo Chaves  Nuno Santos  Gil Dias  Ricardo Pereira  Nuno Nogueira | Identity Module, IdP Proxy, Runtime Registry, Policy Engine, Domain Registry Connector, Registry Data Object Factory, Tutorials |
|  |  |  |
|  |  |  |

# Table of Contents

[Executive summary 4](#_Toc444341024)

[List of authors 5](#_Toc444341025)

[Table of Contents 7](#_Toc444341026)

[Abbreviations 16](#_Toc444341027)

[1 Introduction 18](#_Toc444341028)

[1.1 Objectives and Overview 18](#_Toc444341029)

[1.2 Structure 19](#_Toc444341030)

[2 Tutorials 20](#_Toc444341031)

[2.1 Getting Started 20](#_Toc444341032)

[2.2 Hyper-linked Entities - Hyperties 20](#_Toc444341033)

[2.3 Hyperty Messaging Framework 23](#_Toc444341034)

[2.3.1 Protocol on-the-fly (protofly) and Protostubs 25](#_Toc444341035)

[2.3.2 Message Delivery between different Hyperty Runtimes 26](#_Toc444341036)

[2.4 P2P Data Synchronisation: Reporter - Observer Model 26](#_Toc444341037)

[2.4.1 Hyperty Data Object URL address 27](#_Toc444341038)

[2.4.2 Hyperty Data Object Schema 27](#_Toc444341039)

[2.4.3 Parent - Children Resources 27](#_Toc444341040)

[2.4.4 Syncher and Sync Manager 28](#_Toc444341041)

[2.5 Hyperty Trust and Security Model 29](#_Toc444341042)

[2.5.1 User Identity 30](#_Toc444341043)

[2.5.2 Identity Module and IdP Proxy 30](#_Toc444341044)

[2.5.3 Runtime Sandbox 31](#_Toc444341045)

[2.6 Hyperty Development 32](#_Toc444341046)

[2.6.1 Hyperty Concept 32](#_Toc444341047)

[2.6.2 Criteria to use the Hyperty Concept 32](#_Toc444341048)

[2.6.3 Getting Started 33](#_Toc444341049)

[2.7 Application Development 34](#_Toc444341050)

[2.7.1 Application vs Hyperty 34](#_Toc444341051)

[2.7.2 How to use Hyperties 35](#_Toc444341052)

[2.7.3 How to adapt existing Applications 35](#_Toc444341053)

[2.8 Message Node and Protostubs Development 39](#_Toc444341054)

[2.8.1 Overview 39](#_Toc444341055)

[2.8.2 Messaging Model 41](#_Toc444341056)

[2.8.3 APIs 42](#_Toc444341057)

[2.8.4 Message Node functionalities and main procedures 44](#_Toc444341058)

[2.8.5 Message routing procedure 47](#_Toc444341059)

[2.8.6 Protostub Source Code Examples 48](#_Toc444341060)

[3 Specification changes 51](#_Toc444341061)

[3.1 Hyperty Runtime 51](#_Toc444341062)

[3.2 Messaging Node 52](#_Toc444341063)

[4 References 54](#_Toc444341064)

[Annex A Components Documentation 55](#_Toc444341065)

[A.1 Hyperty Runtime Core 55](#_Toc444341066)

[A.1.1 Overview 55](#_Toc444341067)

[A.1.2 User View 55](#_Toc444341068)

[A.1.3 Developer view 56](#_Toc444341069)

[A.1.4 Example 59](#_Toc444341070)

[A.1.5 Notes 59](#_Toc444341071)

[A.2 dev-runtime-browser 59](#_Toc444341072)

[A.2.1 Overview 59](#_Toc444341073)

[A.2.2 User view 59](#_Toc444341074)

[A.2.3 Developer view 60](#_Toc444341075)

[A.3 Standalone runtime application 61](#_Toc444341076)

[A.3.1 Introduction 61](#_Toc444341077)

[A.3.2 Crosswalk Project 62](#_Toc444341078)

[A.3.3 Android standalone application 63](#_Toc444341079)

[A.3.4 Building the reTHINK Android application 63](#_Toc444341080)

[A.3.5 iOS standalone application 64](#_Toc444341081)

[A.4 vertx.io based Message Node (VertxMN) 66](#_Toc444341082)

[A.4.1 Overview 66](#_Toc444341083)

[A.4.2 User View 66](#_Toc444341084)

[A.4.3 Developer view 67](#_Toc444341085)

[A.5 Matrix.org based Message Node (MatrixMN) 69](#_Toc444341086)

[A.5.1 Overview 69](#_Toc444341087)

[A.5.2 User View 69](#_Toc444341088)

[A.5.3 Developer view 71](#_Toc444341089)

[A.5.4 Matrix.org - Overview and core concepts 72](#_Toc444341090)

[A.6 NodeJS based Message Node 78](#_Toc444341091)

[A.6.1 Overview 78](#_Toc444341092)

[A.6.2 User View 78](#_Toc444341093)

[A.6.3 Developer View 80](#_Toc444341094)

[Annex B APIs Documentation 84](#_Toc444341095)

[B.1 AddressAllocation 84](#_Toc444341096)

[B.1.1 Constructor Summary 84](#_Toc444341097)

[B.1.2 Member Summary 84](#_Toc444341098)

[B.1.3 Method Summary 84](#_Toc444341099)

[B.1.4 Public Constructors 84](#_Toc444341100)

[B.1.4.1 public constructor(url: URL.URL, bus: MiniBus) source 84](#_Toc444341101)

[B.1.5 Public Members 84](#_Toc444341102)

[B.1.5.1 public get url: string: \* source 84](#_Toc444341103)

[B.1.6 Public Methods 84](#_Toc444341104)

[B.1.6.1 public create(domain: Domain, number: number): Promise<HypertyURL> source 84](#_Toc444341105)

[B.2 Bus 85](#_Toc444341106)

[B.2.2 Constructor Summary 85](#_Toc444341107)

[B.2.3 Method Summary 85](#_Toc444341108)

[B.2.4 Public Constructors 86](#_Toc444341109)

[B.2.4.1 public constructor source 86](#_Toc444341110)

[B.2.5 Public Methods 86](#_Toc444341111)

[B.2.5.1 public addListener(url: URL, listener: Listener): MsgListener source 86](#_Toc444341112)

[B.2.5.2 public addResponseListener(url: URL, msgId: number, responseListener: Function) source 86](#_Toc444341113)

[B.2.5.3 public bind(outUrl: URL, inUrl: URL, target: MiniBus): Bound source 86](#_Toc444341114)

[B.2.5.4 public postMessage(msg: Message, responseCallback: Function): number source 86](#_Toc444341115)

[B.2.5.5 public removeAllListenersOf(url: URL) source 87](#_Toc444341116)

[B.2.5.6 public removeResponseListener(url: URL, msgId: number) source 87](#_Toc444341117)

[B.3 GlobalRegistryRecord 87](#_Toc444341118)

[B.3.1 Constructor Summary 87](#_Toc444341119)

[B.3.2 Member Summary 87](#_Toc444341120)

[B.3.3 Method Summary 87](#_Toc444341121)

[B.3.4 Public Constructors 88](#_Toc444341122)

[B.3.4.1 public constructor source 88](#_Toc444341123)

[B.3.5 Public Members 88](#_Toc444341124)

[B.3.5.1 public userIDs: \* source 88](#_Toc444341125)

[B.3.6 Public Methods 88](#_Toc444341126)

[B.3.6.1 public getRecord(): Object source 88](#_Toc444341127)

[B.4 GraphConnector 88](#_Toc444341128)

[B.4.1 Constructor Summary 88](#_Toc444341129)

[B.4.2 Member Summary 88](#_Toc444341130)

[B.4.3 Method Summary 89](#_Toc444341131)

[B.4.4 Public Constructors 90](#_Toc444341132)

[B.4.4.1 public constructor(HypertyRuntimeURL: string, MessageBus: messageBus) source 90](#_Toc444341133)

[B.4.5 Public Members 90](#_Toc444341134)

[B.4.5.1 public contacts: \* source 90](#_Toc444341135)

[B.4.5.2 public contactsBloomFilter1Hop: \* source 90](#_Toc444341136)

[B.4.5.3 public globalRegistryRecord: \* source 90](#_Toc444341137)

[B.4.5.4 public groups: \* source 90](#_Toc444341138)

[B.4.5.5 public lastCalculationBloomFilter1Hop: \* source 90](#_Toc444341139)

[B.4.5.6 public get messageBus(messageBus: MessageBus): \* source 90](#_Toc444341140)

[B.4.5.7 public set messageBus(messageBus: MessageBus): \* source 90](#_Toc444341141)

[B.4.6 Public Methods 90](#_Toc444341142)

[B.4.6.1 public addContact(guid: string, firstName: string, lastname: string) source 90](#_Toc444341143)

[B.4.6.2 public addUserID(userID: string) source 90](#_Toc444341144)

[B.4.6.3 public calculateBloomFilter1Hop() source 91](#_Toc444341145)

[B.4.6.4 public checkGUID(guid: string): array source 91](#_Toc444341146)

[B.4.6.5 public generateGUID(): string source 91](#_Toc444341147)

[B.4.6.6 public getContact(name: string): array source 91](#_Toc444341148)

[B.4.6.7 public queryGlobalRegistry(guid: string): Promise source 91](#_Toc444341149)

[B.4.6.8 public removeContact(guid: string) source 92](#_Toc444341150)

[B.4.6.9 public removeUserID(userID: string) source 92](#_Toc444341151)

[B.4.6.10 public sendGlobalRegistryRecord(jwt: string): Propmise source 92](#_Toc444341152)

[B.4.6.11 public signGlobalRegistryRecord(): string source 92](#_Toc444341153)

[B.4.6.12 public useGUID(mnemonicAndSalt: string): Promise source 92](#_Toc444341154)

[B.5 GraphConnectorContactData 92](#_Toc444341155)

[B.5.1 Constructor Summary 92](#_Toc444341156)

[B.5.2 Member Summary 93](#_Toc444341157)

[B.5.3 Public Constructors 94](#_Toc444341158)

[B.5.3.1 public constructor(guid: String, firstName: String, lastName: String) source 94](#_Toc444341159)

[B.5.4 Public Members 94](#_Toc444341160)

[B.5.4.1 public set contactsBloomFilter1Hop(bf: BloomFilter): \* source 94](#_Toc444341161)

[B.5.4.2 public get contactsBloomFilter1Hop: BloomFilter: \* source 94](#_Toc444341162)

[B.5.4.3 public get firstName: String: \* source 94](#_Toc444341163)

[B.5.4.4 public set firstName(firstName: String): \* source 95](#_Toc444341164)

[B.5.4.5 public set groups(groups: List<String>): \* source 95](#_Toc444341165)

[B.5.4.6 public get groups: List<String>: \* source 95](#_Toc444341166)

[B.5.4.7 public get guid: String: \* source 95](#_Toc444341167)

[B.5.4.8 public set guid(GUID: String): \* source 95](#_Toc444341168)

[B.5.4.9 public set lastName(lastName: String): \* source 95](#_Toc444341169)

[B.5.4.10 public get lastName: String: \* source 95](#_Toc444341170)

[B.5.4.11 public set privateContact(boolPrivate: Boolean): \* source 95](#_Toc444341171)

[B.5.4.12 public get privateContact: Boolean: \* source 95](#_Toc444341172)

[B.5.4.13 public get residenceLocation: String: \* source 96](#_Toc444341173)

[B.5.4.14 public set residenceLocation(geohash: String): \* source 96](#_Toc444341174)

[B.5.4.15 public set userIDs(userIDs: List<String>): \* source 96](#_Toc444341175)

[B.5.4.16 public get userIDs: List<String>: \* source 96](#_Toc444341176)

[B.6 HypertyDiscovery 96](#_Toc444341177)

[B.6.1 Constructor Summary 96](#_Toc444341178)

[B.6.2 Method Summary 96](#_Toc444341179)

[B.6.3 Public Constructors 96](#_Toc444341180)

[B.6.3.1 public constructor(msgbus: MessageBus, runtimeURL: RuntimeURL) source 96](#_Toc444341181)

[B.6.4 Public Methods 97](#_Toc444341182)

[B.6.4.1 public discoverHypertyPerUser(email: email): Promise source 97](#_Toc444341183)

[B.7 IdentityModule 97](#_Toc444341184)

[B.7.1 Constructor Summary 97](#_Toc444341185)

[B.7.2 Method Summary 98](#_Toc444341186)

[B.7.3 Public Constructors 98](#_Toc444341187)

[B.7.3.1 public constructor source 98](#_Toc444341188)

[B.7.4 Public Methods 98](#_Toc444341189)

[B.7.4.1 public generateAssertion(contents: DOMString, origin: DOMString, usernameHint: DOMString): IdAssertion source 98](#_Toc444341190)

[B.7.4.2 public getAssertionTrustLevel(assertion: DOMString) source 99](#_Toc444341191)

[B.7.4.3 public getIdentities(): Array<Identities> source 99](#_Toc444341192)

[B.7.4.4 public loginWithRP(identifier: Identifier, scope: Scope): Promise source 99](#_Toc444341193)

[B.7.4.5 public registerIdentity() source 99](#_Toc444341194)

[B.7.4.6 public registerWithRP() source 99](#_Toc444341195)

[B.7.4.7 public setHypertyIdentity() source 99](#_Toc444341196)

[B.7.4.8 public validateAssertion(assertion: DOMString): Promise source 100](#_Toc444341197)

[B.8 MessageBus 100](#_Toc444341198)

[B.8.2 Constructor Summary 100](#_Toc444341199)

[B.8.3 Member Summary 100](#_Toc444341200)

[B.8.4 Method Summary 100](#_Toc444341201)

[B.8.5 Inherited Summary 100](#_Toc444341202)

[B.8.6 Public Constructors 101](#_Toc444341203)

[B.8.6.1 public constructor(registry: \*) source 101](#_Toc444341204)

[B.8.7 Public Members 101](#_Toc444341205)

[B.8.7.1 public get pipeline: \* source 101](#_Toc444341206)

[B.8.8 Public Methods 101](#_Toc444341207)

[B.8.8.1 public addForward(from: \*, to: \*): \* source 101](#_Toc444341208)

[B.8.8.2 public postMessage(inMsg: \*, responseCallback: \*): \* source 101](#_Toc444341209)

[B.9 MiniBus 102](#_Toc444341210)

[B.9.2 Constructor Summary 102](#_Toc444341211)

[B.9.3 Method Summary 102](#_Toc444341212)

[B.9.4 Inherited Summary 102](#_Toc444341213)

[B.9.5 Public Constructors 103](#_Toc444341214)

[B.9.5.1 public constructor source 103](#_Toc444341215)

[B.9.6 Public Methods 103](#_Toc444341216)

[B.9.6.1 public postMessage(inMsg: \*, responseCallback: \*): \* source 103](#_Toc444341217)

[B.10 ObjectAllocation 103](#_Toc444341218)

[B.10.1 Constructor Summary 103](#_Toc444341219)

[B.10.2 Member Summary 103](#_Toc444341220)

[B.10.3 Method Summary 103](#_Toc444341221)

[B.10.4 Public Constructors 103](#_Toc444341222)

[B.10.4.1 public constructor(url: URL.URL, bus: MiniBus) source 103](#_Toc444341223)

[B.10.5 Public Members 104](#_Toc444341224)

[B.10.5.1 public get url: string: \* source 104](#_Toc444341225)

[B.10.6 Public Methods 104](#_Toc444341226)

[B.10.6.1 public create(domain: Domain, number: number): Promise<ObjectURL> source 104](#_Toc444341227)

[B.11 Pipeline 104](#_Toc444341228)

[B.11.1 Constructor Summary 104](#_Toc444341229)

[B.11.2 Method Summary 104](#_Toc444341230)

[B.11.3 Public Constructors 104](#_Toc444341231)

[B.11.3.1 public constructor(\_onFail: \*) source 104](#_Toc444341232)

[B.11.4 Public Methods 105](#_Toc444341233)

[B.11.4.1 public process(msg: \*, onDeliver: \*) source 105](#_Toc444341234)

[B.12 PolicyEngine 105](#_Toc444341235)

[B.12.1 Constructor Summary 105](#_Toc444341236)

[B.12.2 Method Summary 105](#_Toc444341237)

[B.12.3 Public Constructors 105](#_Toc444341238)

[B.12.3.1 public constructor(identityModule: \*, runtimeRegistry: \*) source 105](#_Toc444341239)

[B.12.4 Public Methods 105](#_Toc444341240)

[B.12.4.1 public addPolicies(key: \*, policies: \*) source 105](#_Toc444341241)

[B.12.4.2 public authorise(message: \*): \* source 106](#_Toc444341242)

[B.12.4.3 public checkPolicies(message: \*, userID: \*): \* source 106](#_Toc444341243)

[B.12.4.4 public getApplicablePolicies(hypertyFrom: \*, hypertyTo: \*, userID: \*): \* source 106](#_Toc444341244)

[B.12.4.5 public getAuthorisationDecision(results: \*): \* source 106](#_Toc444341245)

[B.12.4.6 public removePolicies(key: \*, policyId: \*) source 106](#_Toc444341246)

[B.12.4.7 public simulate(key: \*) source 106](#_Toc444341247)

[B.13 Registry 107](#_Toc444341248)

[B.13.2 Constructor Summary 107](#_Toc444341249)

[B.13.3 Member Summary 107](#_Toc444341250)

[B.13.4 Method Summary 107](#_Toc444341251)

[B.13.5 Inherited Summary 108](#_Toc444341252)

[B.13.6 Public Constructors 109](#_Toc444341253)

[B.13.6.1 public constructor(msgbus: MessageBus, runtimeURL: HypertyRuntimeURL, appSandbox: AppSandbox, remoteRegistry: DomainURL) source 109](#_Toc444341254)

[B.13.7 Public Members 109](#_Toc444341255)

[B.13.7.1 public get messageBus(messageBus: MessageBus): \* source 109](#_Toc444341256)

[B.13.7.2 public set messageBus(messageBus: MessageBus): \* source 109](#_Toc444341257)

[B.13.8 Public Methods 109](#_Toc444341258)

[B.13.8.1 public discoverProtostub(DomainURL: DomainURL): RuntimeURL source 109](#_Toc444341259)

[B.13.8.2 public getAppSandbox(): \* source 109](#_Toc444341260)

[B.13.8.3 public getSandbox(DomainURL: DomainURL): RuntimeSandbox source 110](#_Toc444341261)

[B.13.8.4 public getUserHyperty(email: \*): \* source 110](#_Toc444341262)

[B.13.8.5 public onEvent() source 110](#_Toc444341263)

[B.13.8.6 public registerHyperty(sandbox: Sandbox, HypertyCatalogueURL: HypertyCatalogueURL): HypertyURL source 110](#_Toc444341264)

[B.13.8.7 public registerPEP(postMessage: Message.Message, HypertyURL: HypertyURL): HypertyRuntimeURL source 110](#_Toc444341265)

[B.13.8.8 public registerStub(Sandbox: Sandbox, DomainURL: DomainURL): RuntimeProtoStubURL source 111](#_Toc444341266)

[B.13.8.9 public resolve(url: URL.URL): Promise<URL.URL> source 111](#_Toc444341267)

[B.13.8.10 public unregisterHyperty(HypertyURL: HypertyURL): \* source 111](#_Toc444341268)

[B.13.8.11 public unregisterPEP(HypertyRuntimeURL: HypertyRuntimeURL): \* source 111](#_Toc444341269)

[B.13.8.12 public unregisterStub(HypertyRuntimeURL: HypertyRuntimeURL): \* source 112](#_Toc444341270)

[B.14 RuntimeCatalogue 112](#_Toc444341271)

[B.14.1 Constructor Summary 112](#_Toc444341272)

[B.14.2 Member Summary 112](#_Toc444341273)

[B.14.3 Method Summary 112](#_Toc444341274)

[B.14.4 Public Constructors 113](#_Toc444341275)

[B.14.4.1 public constructor(nodeHttp: \*, nodeHttps: \*) source 113](#_Toc444341276)

[B.14.5 Public Members 113](#_Toc444341277)

[B.14.5.1 public set runtimeURL: \* source 113](#_Toc444341278)

[B.14.5.2 public get runtimeURL: \* source 113](#_Toc444341279)

[B.14.6 Public Methods 113](#_Toc444341280)

[B.14.6.1 public getDataSchemaDescriptor(dataSchemaURL: \*): Promise source 113](#_Toc444341281)

[B.14.6.2 public getHypertyDescriptor(hypertyURL: \*): Promise source 113](#_Toc444341282)

[B.14.6.3 public getHypertyRuntimeURL(): \* source 114](#_Toc444341283)

[B.14.6.4 public getRuntimeDescriptor(runtimeURL: \*): Promise source 114](#_Toc444341284)

[B.14.6.5 public getSourceCodeFromDescriptor(descriptor: \*): Promise source 114](#_Toc444341285)

[B.14.6.6 public getSourcePackageFromURL(sourcePackageURL: \*): Promise source 114](#_Toc444341286)

[B.14.6.7 public getStubDescriptor(stubURL: \*): Promise source 114](#_Toc444341287)

[B.15 RuntimeUA 115](#_Toc444341288)

[B.15.1 Constructor Summary 115](#_Toc444341289)

[B.15.2 Method Summary 115](#_Toc444341290)

[B.15.3 Public Constructors 115](#_Toc444341291)

[B.15.3.1 public constructor(sandboxFactory: sandboxFactory, domainURL: domain) source 115](#_Toc444341292)

[B.15.4 Public Methods 116](#_Toc444341293)

[B.15.4.1 public checkForUpdate(url: CatalogueURL) source 116](#_Toc444341294)

[B.15.4.2 public discoverHiperty(descriptor: CatalogueDataObject.HypertyDescriptor) source 116](#_Toc444341295)

[B.15.4.3 public loadHyperty(hyperty: URL.HypertyCatalogueURL): \* source 116](#_Toc444341296)

[B.15.4.4 public loadStub(domain: URL.URL): \* source 116](#_Toc444341297)

[B.15.4.5 public registerHyperty(Object: Object, descriptor: URL.HypertyCatalogueURL) source 116](#_Toc444341298)

[B.16 Sandbox 117](#_Toc444341299)

[B.16.2 Constructor Summary 117](#_Toc444341300)

[B.16.3 Method Summary 117](#_Toc444341301)

[B.16.4 Inherited Summary 117](#_Toc444341302)

[B.16.5 Public Constructors 118](#_Toc444341303)

[B.16.5.1 public constructor source 118](#_Toc444341304)

[B.16.6 Public Methods 118](#_Toc444341305)

[B.16.6.1 public deployComponent(componentSourceCode: string, componentURL: URL, configuration: Config): Promise<string> source 118](#_Toc444341306)

[B.16.6.2 public removeComponent(componentURL: URL): Promise<string> source 118](#_Toc444341307)

[B.17 SyncherManager 118](#_Toc444341308)

[B.17.1 Constructor Summary 118](#_Toc444341309)

[B.17.2 Member Summary 118](#_Toc444341310)

[B.17.3 Public Constructors 119](#_Toc444341311)

[B.17.3.1 public constructor(runtimeURL: \*, bus: \*, registry: \*, catalog: \*, allocator: \*) source 119](#_Toc444341312)

[B.17.4 Public Members 119](#_Toc444341313)

[B.17.4.1 public get url: \* source 119](#_Toc444341314)

# Abbreviations

|  |  |
| --- | --- |
| API | Application Programming Interface |
| COAP | Constrained Application Protocol |
| CRUD | Create, Retrieve, Update and Delete |
| CSP | Communication service provider |
| DDoS | Distributed Denial of Service Attacks |
| DoS | Denial of Service |
| H2H | Human to Human communication |
| ICE | Information and Content Exchange |
| IETF | Internet Engineering Task Force |
| JSON | JavaScript Object Notation |
| LWM2M | LightweightM2M |
| M2M | Machine to Machine communication |
| ORTC | Object Real-Time Communications |
| QoS | Quality of Service |
| REST | Representational State Transfer |
| STUN | Session Traversal Utilities for NAT |
| TURN | Traversal Using Relay NAT |
| UML | Unified Modelling Language |
| URI | Uniform Resource Identifier |
| URL | Uniform Resource Locator |
| W3C | World Wide Web Consortium |
| WHATWG | Web Hypertext Application Technology Working Group |
| SPPE | Service Provider Policy Enforcer |
| PEE | Policy Enforcer Engine |
| TRAM | TURN Revised and Modernized |
| HTTP | Hypertext Transfer Protocol |
| TCP | Transmission Control Protocol |
| QUIC | Quick UDP Internet Connections |
| XMPP | Extensible Messaging and Presence Protocol |
| ORTC | Object Real-Time Communications |
| COAP | Constrained Application Protocol |
| LWM2M | Lightweight M2M |
| SDT | Smart Device Template |
| HGI | Home Gateway Iniative |
| SFU | Selective Forwarding Unit |
| MCU | Multipoint Control Unit |
| TLS | Transport Layer Security |
| MQTT | MQ Telemetry Transport |
| WAC | WebRTC Application Controller |
| AAA | Authentication, Authorization and Accounting |
| OSS | Operations Support System |
| BSS | [business support systems](https://en.wikipedia.org/wiki/Business_support_system) |
| RCS | Rich Communication Services |
| UC | Unified Communications |
| CRM | [Customer Relationship Management](https://en.wikipedia.org/wiki/Customer_relationship_management) |
| JSONoWS | JSON over Web Sockets |
| IdP | Identity Provider |
| TCB | Trusted Computing Base |
| PDP | Policy Decision Point |
| PEP | Policy Enforcement Point |

# Introduction

## Objectives and Overview

Project reTHINK aims to demonstrate a radical new way to develop and deliver real time communication services. reTHINK concepts and architecture follows edge computing principles [18] pushing as much as possible the Business logic to end-user devices and network edge servers. reTHINK Business Capabilities are provided by cooperative Microservices [17] executing on behalf of users, called Hyperlinked Entities, named Hyperties. Hyperties are independently deployable components, each one providing a small set of business capabilities, using the smart endpoints and dumb pipes philosophy. Billions of devices are already Hyperty-enabled and ready to participate in a reTHINK ecosystem, since Hyperties are programmed in Javascript ECMA5/6. The User Identity associated to a Hyperty instance is decoupled from the Hyperty Service Provider through WebRTC Identity Management mechanisms.



Figure - Core Framework Components scope in reTHINK Architecture

This deliverable accompanies the initial release of the Core Framework components published in reTHINK Github repositories [1]. The implementation of Core Framework components follows D2.1 [2] reTHINK Architecture design, D2.2 reTHINK Data Models [3] and D3.1 Core Framework detailed specification [4]. Phase 2 of the Core Framework includes the Hyperty Runtime (Core Runtime Components [7] that were reused in Hyperty Browser Runtime [8] and Hyperty NodeJS Runtime[9]) and three Message Node implementations: Vertx Message Node[10], Matrix Message Node[11] and NodeJS Message Node[12]. Finally, the Hyperty Service Framework [13] is also released featuring a comprehensive set of application program interfaces (APIs) and JavaScript libraries to facilitate the development of Hyperties. A full suite of documentation specially written to facilitate reTHINK embracing by web developers, which is also published in GitHub pages, is included in this report, as well as a short summary of major changes done in the specification of reTHINK Core Framework components provided in D3.1, according to feedback taken from the implementation activity.

It should be noted that the Network Platform components supporting Specialised Network Services will be released later in D3.4, as originally planned. This means, Use Cases featuring QoS will only be released in Phase 2. The final specification for Messaging Node and Hyperty Runtime will be reported in D3.3 (Hyperty Runtime and Hyperty Messaging Node Phase 2 – Dec 2016).

This deliverable complements deliverable D4.2 (Management and Security features implementation [6]), which accompanies Phase 2 release of reTHINK Support Services.

## Structure

This report starts with an introduction followed by a set of Tutorials (Chapter 2) targeting reTHINK Web Developers are provided. In Chapter 3, a summary of major changes done in the original specification, is given. The Documentation of Phase 2 Core Framework components is provided as annexes including API documentation.

# Tutorials

This chapter provides the full set of documentation targeting external developers that is published in reTHINK Github together with the Hyperty Service Framework source code. These documents will be updated and completed along the implementation of Phase 2 demonstration scenarios. A getting started document is provided followed by an overview of the Hyperty Concept, the Messaging Framework used to support Hyperty's interaction, the higher level Data Synchronisation Reporter - Observer communication mechanism, as well as the Hyperty Trust and Security Model. Then, detailed Tutorials are provided to guide on the development of Hyperties, Applications and Message Nodes.

## Getting Started

The reTHINK Framework is built around Hyperlinked Entities (hyperties) that leverages the protocol-on-the-fly (ProtoFly) concept to avoid creating or modifying standard network protocols, but moving instead towards an API based flat service architecture. The reTHINK project envisages a global network of interconnected hyperties that are executed in web runtime environment on endpoints or edge-network servers. The reTHINK Framework functions are based on a series of such hyperties that are generated by the service provider, and are downloaded to the users’ endpoints. The hyperty modules represent a set of services that are stored in a Catalogue. The instantiated versions of these hyperties are registered in a Registry, which represents authenticated users who are available for in-coming connectivity service. Therefore, the Registry serves as a location manager and is used for user discovery. A user can be a human being, a group or, a connected object (e.g. building room with sensors). Users have independent identities that are maintained by Identity Providers (IdPs) that are independent organizations. On enquiry, these IdPs vouch for users. The users’ identities are based on their personal and confidential data, which is verified by other solicited data, but such private information is only divulged under user-controlled privacy rules.

In order to setup your own reTHINK Framework, you should install the following components (docker images available):

* the [Hyperty Catalogue](https://github.com/reTHINK-project/dev-catalogue)[15]
* the [Hyperty Domain Registry](https://github.com/reTHINK-project/dev-registry-domain)[16]
* any of the three available Message Nodes: [Vertx Message Node](https://github.com/reTHINK-project/dev-msg-node-vertx) [10], [Matrix](https://matrix.org/) Message Node [11] or [NodeJS](https://nodejs.org/en/) Message Node [12].

Very soon a live public reTHINK environment will be provided, to let you publish and try your Hyperties or Apps without the need to install anything.

## Hyper-linked Entities - Hyperties

This document provides an overview about the Hyperty concept and it should be the starting point for any new developer. After this document, all developers should also read:

* the Hyperty Messaging Framework overview
* the Reporter - Observer Data Synchronisation model
* the Hyperty Trust and Security Model

Hyperties are cooperative [Microservices](http://martinfowler.com/articles/microservices.html) [17] that are executed in devices on behalf of users through simple but sophisticated Identity Management techniques. This means, Hyperties are independently deployable components each one providing a small set of business capabilities, using the *smart endpoints and dumb pipes* philosophy i.e. Hyperties don't depend on complex and sophisticated communication middleware like Enterprise Service BUS (ESB). Instead, Hyperties rely on a very light but powerful Messaging Framework concept). On the other side, Hyperties follow emerging [Edge](https://en.wikipedia.org/wiki/Edge_computing) [18] and [Fog](https://en.wikipedia.org/wiki/Fog_computing)[19] computing paradigms as opposed to more popular Cloud Computing. Hyperties can also be executed in Network Servers for specific Business Capabilities (e.g. Media Servers) or when End-user devices don't have enough capabilities in terms of computing resources and/or power.

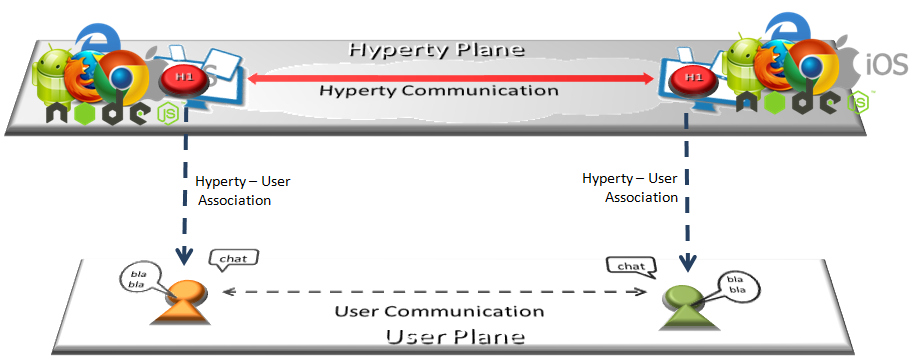


Figure - Hyperty Concept and Edge Computing

Compared with other Real Time Communication Development Frameworks, Hyperties provides a few unique advantages:

* Hyperty reinforces modular structure, which is particularly important for larger and complex Applications.
* Hyperties from different Service Providers can easily cooperate with no federation required or the standardisation of Protocols.
* Hyperties uses a simple but powerful Trust model where the User Identity is decoupled from the Hyperty Service Provider enabling the end-user to decide which is the Identity to be used with a certain Hyperty instance.
* Hyperties follow edge computing principles, promoting a more effective usage of computing and network resources, as well as decreasing communication latency.
* the development of Hyperties and Applications is very easy and flexible, giving the freedom to the developer to select its favorite web framework.
* Hyperties can be used on any Application Domain, but they are specially suitable for Real Time Communication Apps (eg Video Conference and Chat) as well as IoT Apps.

More details about Hyperties unique characteristics:

* Hyperties are programmed in Javascript ECMA5/6, i.e. any existing device featuring a Browser or a NodeJS can be used today to execute Hyperties without requiring the installation of any new software. This means, **billions of devices** are already Hyperty enabled and ready to participate in the reTHINK ecosystem. The [Hyperty Core Runtime](https://github.com/reTHINK-project/dev-runtime-core), provides additional features not natively supported by current Web Runtimes that are required to safely manage the deployment and execution of Hyperties. The Hyperty Core Runtime is also programmed in Javascript ECMA5/6 and is deployed on-the-fly along with the Hyperty if not done before.
* The User Identity associated to an Hyperty is decoupled from the Hyperty Service Provider. I.e. Identity Management is handled under the scene and the Developer does not have to care about it and just have to focus on the development of Business Capabilities. This also means, the end-user has the power to decide which is the Identity to be securely associated to a certain Hyperty instance. More information is provided in the Hyperty Security and Trust Model document.
* Hyperties cooperate and communicate each other via P2P Synchronisation of Hyperty JSON Data Objects supported by the Reporter - Observer communication pattern. For example, as soon as there is new measurement collected from a sensor the data is set in a associated JSON Object. As soon as there is a change in this JSON Object, the change is reported by the Reporter Hyperty to any authorised Observer Hyperty. In this way, the JSON Object handled by Observer Hyperty is always synchronised with the JSON Object owned by the Reporter Hyperty.



Figure - Reporter-Observer Communication Pattern

The API to handle the Synchronisation of Hyperty Data Objects is extremely simple and fun to use. The Developer of the Hyperty Reporter just has to create the Data Sync object with the Syncher API, and write on the object every time there is data to be updated and shared with Hyperty Observers.

syncher.create(\_this.\_objectDescriptorURL, [invitedHypertyURL], hello).then(function(dataObjectReporter){

console.info('Hello Data Object was created: ', dataObjectReporter);

dataObjectReporter.data.hello = "Bye!!";

}

On the Hyperty Observer side, Data Objects are also created with the Syncher API and the emerging [Object.observer() Javascript method](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/observe) is used to receive the stream of data changes coming from the Reporter Hyperty.

syncher.subscribe(\_this.\_objectDescriptorURL, ObjectUrl).then(function(dataObjectObserver) {}  
  
 dataObjectObserver.onChange('\*', function(event) {  
 // Hello World Object was changed  
 console.info(dataObjectObserver);  
 });

* Hyperties can easily cooperate with Hyperties from other domains with no federation required or the standardisation of Protocols thanks to the Protocol On-the Fly powered Messaging Framework. Hyperties only have to agree on a common json-schema for one or more Hyperty Data Objects, in order to be able to cooperate each other.
* Hyperties can be used on any Application Domain, but they are specially suitable for Real Time Communication Apps (eg Video Conference and Chat) as well as IoT Apps.

## Hyperty Messaging Framework

This document gives an overview on the Messaging Framework technical solution used to support Hyperty's interaction through the higher level Data Synchronisation Reporter - Observer communication mechanism. Check the Hyperty Development Tutorial for more details.

Hyperties cooperate each other with a Resource Oriented Messaging model implemented by a simple Messaging Framework. The Hyperty Messaging Framework, supports different messaging patterns including publish/subscribe and request/response messaging patterns. The higher level Reporter - Observer communication pattern works on top of these basic messaging patterns. It should be noted, that Hyperty Service Development Framework to be used to create new Hyperties, abstracts Developers from the Hyperty Messaging Framework described in this document, including lower level Hyperty Messaging APIs.

The Message delivery is based on a simple message Router functionality that performs a lookup for listeners registered to receive the Message (the ["Message.to" Header field](https://github.com/reTHINK-project/dev-service-framework/blob/develop/docs/datamodel/message/readme.md#to) is the only information looked up for). The Message is posted to all found listeners, which can be other Routers or end-points (Hyperties). Thus, the Hyperty Messaging Framework is comprised by a network of Routers where each Router only knows adjacent registered Routers or end-points.



Figure - Hyperty Messaging Delivery Network

Listeners are programmatically registered and unregistered by Routing Management functionalities, which decide the listeners to be added according to a higher level view of the Routing Network.



Figure - Hyperty Message Routing Management

The Messaging Framework works at three layers:

At the Runtime Sandbox level where Hyperties are executing, message delivery is provided by the [MiniBUS component](https://github.com/reTHINK-project/dev-runtime-core/blob/master/src/bus/MiniBus.js).

At the Runtime level where Sandboxes are hosted (e.g. in a Browser or in a NodeJS instance), message delivery is provided by the [Message BUS component](https://github.com/reTHINK-project/dev-runtime-core/blob/master/src/bus/MessageBus.js), which is an extension of the MiniBUS.

At Domain Level, message delivery is provided by the Message Node functionality by using the Protofly mechanism, i.e. communication between Message BUS and Message Nodes and among Message Nodes are protocol agnostic. This also means that the Message Node can be provided by any Messaging solution as soon as there is a Protostub available. Currently, a [Vertx Message Node](https://github.com/reTHINK-project/dev-msg-node-vertx), a [Matrix Message Node](https://github.com/reTHINK-project/dev-msg-node-matrix) and a [NodeJS Message Node](https://github.com/reTHINK-project/dev-msg-node-nodejs) are provided. These are just reference implementations of Message Nodes and anyone is free to develop its own Message Node. Check the Message Node and associated Protostub development tutorial for more details.



Figure - Adhoc Messaging Oriented Middleware Routing Layers

At runtime level (MessageBUS and MiniBUS), it is used a standard CRUD based [JSON Message Model](../datamodel/message/readme.md), which is easily mapped into Restfull APIs.

### Protocol on-the-fly (protofly) and Protostubs

Protocol on-the-fly leverages the code on-demand support by Web runtimes (eg Javascript), to dynamically select, load and instantiate the most appropriate protocol stack during run-time. Such characteristic enables protocols to be selected at run-time and not at design time, enabling protocol interoperability among distributed services, promoting loosely coupled service architectures, optimising resources spent by avoiding the need to have Protocol Gateways in service's middleware as well as minimising standardisation efforts. The implementation of the protocol stack, e.g. in a javascript file, that is dynamically loaded and instantiated in run-time is called **Protostub:**. For security reasons, Protostubs are executed in isolated sandboxes and are only reachable through the Runtime MessageBUS and the Protostub Sandbox MiniBUS.



Figure - Protocol on-the-fly and Protostubs

### Message Delivery between different Hyperty Runtimes

Communication between the Message BUS and Message Nodes are provided by a Protostub that implements the protocol stack used to interact with the Message Node e.g. JSON over Websockets or a Restfull API Client. Listeners of protostubs are registered in the MessageBUS for a set of Message recipient addresses, usually a Hyperty Domain like domain://example.com.

When the MessageBUS is processing a new message and looking up routing paths for an address, which is not local (eg hyperty://example.com/alice-hyperty), it won't find any registered listeners. In this case, the MessageBUS will ask the [Runtime Registry](https://github.com/reTHINK-project/dev-runtime-core/blob/d3.2-working-docs/docs/specs/readme.md#runtime-registry) to resolve the "Message.to" header field, which will look for registered Protostubs that are able to deliver messages to such non-local address. If there is already a deployed Protostub that is able to deliver the message to the remote Hyperty, the Registry will return the Hyperty Runtime protostub address and the MessageBUS will look up again for the protostub listener registered for its address. Otherwise, the deployment of the required Protostub is performed and as soon as the Protostub is successfully instantiated, its hyperty runtime address is returned.

## P2P Data Synchronisation: Reporter - Observer Model

This document gives an overview on how Hyperties cooperate each other through a Data Synchronisation model called Reporter - Observer.

The usage of Data synchronisation models in [Web Frameworks](https://www.meteor.com/ddp) are becoming very popular and the usage of the emerging [object.observe](https://developer.mozilla.org/pt-PT/docs/Web/JavaScript/Reference/Global_Objects/Object/observe) javascript API is making it even more appealing. However, current solutions require server-side databases that are not compliant with edge computing Hyperty principles.

Hyperty Reporter - Observer communication pattern goes beyond current solutions by using a P2P Synchronisation solution for JSON Data Objects, here called Hyperty Data Object or Sync Data Object. To avoid concurrency inconsistencies among peers, only one peer has granted writing permissions in the Hyperty Data Object - the **Reporter hyperty** - and all the other Hyperty instances only have permissions to read the Hyperty Data Object - the **Observer hyperty**.



Figure - Reporter-Observer Communication Pattern

The API to handle Hyperty Data Objects is extremely simple and fun to use. The Developer of the Hyperty Reporter just has to create the Data Sync object with the Syncher API,

syncher.create(\_this.\_objectDescriptorURL, [invitedHypertyURL], hello).then(function(dataObjectReporter){  
  
 console.info('Hello Data Object was created: ', dataObjectReporter);

}

... and write on the object every time there is data to be updated and shared with Hyperty Observers:

dataObjectReporter.data.hello = "Bye!!";

On the Hyperty Observer side, Data Objects are subscribed with the Syncher API,

syncher.subscribe(\_this.\_objectDescriptorURL, ObjectUrl).then(function(dataObjectObserver) {}  
  
 console.info( dataObjectObserver);

... and the emerging [Object.observer() Javascript method](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/observe) is used to receive the stream of data changes coming from the Reporter Hyperty.

dataObjectObserver.onChange('\*', function(event) {  
 // Hello World Object was changed  
 console.info(dataObjectObserver);  
 });

### Hyperty Data Object URL address

The Hyperty Messaging Framework allocates to each new created Hyperty Data Object a Global Unique Identifier URL that is independent from the Hyperty instance creator and from the Hyperty Runtime, in order to support mobility of the Data Object between different Hyperty Runtimes and also to support delegation of the Reporter role to other Hyperty instances. However, at this point Reporter delegation is only supported between Hyperty instances from the same domain.

### Hyperty Data Object Schema

Each Hyperty Data Object is formally described by a JSON-Schema that is identified by a Catalogue URL. This allows to check whether two different Hyperties are compliant by cross checking each supported Hyperty Data Object schema. At this point the following Hyperty Data Object schemas are defined:

* [**Connection Data Schema**](../datamodel/connection) : Hyperties supporting this schema are able to handle [WebRTC Peer Connections](https://developer.mozilla.org/en-US/docs/Web/Guide/API/WebRTC/Peer-to-peer_communications_with_WebRTC) between the Hyperty Runtime instances where they are running independently of the signalling protocol used. The URL Scheme for Connection Data Objects is "connection" (example: "connection://example.com/alice/bob201601290617").
* [**Communication Data Schema**](../datamodel/communication) : Hyperties supporting this schema are able to handle different communication types including Textual Chat, Audio, Video, Screen Sharing and File sharing. Such communication can be supported on top of WebRTC protocol streams by using the Connection Data Schema. The URL Scheme for Communication Data Objects is "comm" (example: "comm://example.com/group-chat/rethink201601290617").
* [**Context Data Schema**](../datamodel/context) : Hyperties supporting this schema are able to produce or consume Context Data, usually collected from sensors. The URL Scheme for Communication Data Objects is "ctxt" (example: "ctxt://example.com/room/temperature201601290617").

### Parent - Children Resources

In order to allow use cases like Group Chat where all involved Hyperties are able to write in the Sync Data Object, the Parent - Child Data Sync Objects is introduced.

A Data Object Child belongs to a Data Object Parent children resource and can be created by any Observer of the Data Object Parent as well as by its Reporter. The Reporter - Observer rules still apply to Data Object Child i.e. there is only one Reporter that can update the Data Object Child, which can be an Observer of the Data Object Parent, as mentioned earlier.



Figure - Parent - Child Sync

The creation, update and delete of an Data Object Child is performed in the Data Object Parent itself.

All other Hyperties observing or reporting the Data Object Parent, will be notified every time a new Data Object Child is created, updated or deleted:

At this point, Data Object Child can't also be a Data Object Parent of another Sync Data Object, i.e. Hyperty Data Object composition is limited to one level.

### Syncher and Sync Manager

This section, gives an overview on how the Hyperty Data Object synchronisation transparently works on top of the Hyperty Messaging Middleware. However, Hyperty developers don't have to know the technical details of this solution and can directly move to the Hyperty Development Manual.

The Hyperty Data Object synchronisation is provided by two components in the Runtime:

The [Syncher](https://github.com/reTHINK-project/dev-service-framework/blob/master/src/syncher/Syncher.js) is a singleton Component co-located with the Hyperty Instance, which is in charge of handling all required procedures to manage data synchronisation at the Hyperty instance side, as a Reporter or a Observer Hyperty.

The [Runtime Sync Manager](https://github.com/reTHINK-project/dev-service-framework/blob/master/src/syncher/Syncher.js) is a Core Runtime Component, which is in charge of handling authorisation requests to create Sync Data Objects from Hyperty Reporters and subscription requests to Sync Data Objects from Hyperty Observers. As soon as authorisation is granted the Sync Manager handles all required MessageBUS listeners in order to setup the Data Sync Stream routing path among Reporters and Observers. I.e., the Runtime Sync Manager provides a Messaging Middleware Routing Manager functionality.

The [Message Node Sync Manager](https://github.com/reTHINK-project/dev-service-framework/blob/master/src/syncher/Syncher.js) is a Message Node functionality, which is in charge of handling requests from Runtime Sync Managers in order to setup the Data Sync Stream routing path between the Reporter Hyperty Runtime and Observers Hyperty Runtimes. I.e., the Message Node Sync Manager also provides a [Messaging Middleware](hyperty-messaging-middleware.md) Routing Manager functionality..



Figure - Routing Management for Hyperty Data Syncronisation

## Hyperty Trust and Security Model

This document gives an overview on the Hyperty Trust Model as well as on Hyperty Sandbox runtime execution environment.

It should be noted, that Hyperty Service Development Framework to be used to create new Hyperties, abstracts Developers from the Hyperty Trust and Security Model described in this document including lower level Identity Management APIs. Details about how to develop Hyperties is provided in [this](development-of-hyperties.md) document.

Hyperties are securely associated to User Identities selected by the end-user himself. Hyperty Users are human beings (including group of human beings e.g. corporation) or things (including group of things and physical spaces e.g. a smart home or smart building).

Hyperty Trust Model extends [WebRTC Identity model](https://w3c.github.io/webrtc-pc/#sec.identity-proxy) where Identity tokens are generated, inserted in intercepted Messages sent by Hyperties and validated by recipient Hyperty Runtimes before delivered to the target Identity. These identity management procedures are performed according to applicable policies managed by the end-user.



Figure - Hyperty Trust Management

### User Identity

In our modern society, technology is ubiquitous, and transactions are evermore accomplished using digital technologies without the need to involve physical contact. An example of this situation can be observed in money transactions, whilst a few years ago if someone needed to make a bank transfer, it would require that person to move personally into a bank agency to order it, and in current days these money transfers can be performed using a smartphone. To achieve this, we need digital credentials to prove who we are and what we are allowed to do in remote communication. Therefore two important information security mechanisms must be implemented: authentication and authorization.

Authentication is verifying the identity claimed by an actor. Usually, to authenticate users, credentials make use of one or several factors among:

* something a user knows (such as a PIN code or a password),
* something a user owns (such as a SIM card),
* something a user is (such as a fingerprint, or a voice sample),
* something a user does (such as typing characteristics).

Authorization is deciding whether a given identity may execute or access a certain resource. Access control to a service or system, can be achieved based on access rights or policies that allow or deny a particular action based on an identifier, a role (RBAC), or an attribute (ABAC).

Hyperty trust model relies on service-independent authentication that is global and non-service-bound. In implementations, each Hyperty service provider may include their own user recognition (i.e. their own internal user accounts) and service authorization (i.e. level of permissions to use the service), over and above the initial user identification.

### Identity Module and IdP Proxy

The Identity Module (Id Module) is the [Hyperty Core Runtime](https://github.com/reTHINK-project/dev-runtime-core) component responsible for handling the user identity and the association of this identity with the Hyperty instances, in order to make Hyperty instances identifiable. The identity in the reTHINK project is not fixed to a unique Identity Service Provider, but obtained through several different Identity sources. With this approach, the Id Module provides to the user the option to choose the preferred method for authentication. This module will thus able to support multiple Identity acquisition methods, such as OpenID connect 1.0, Kerberos System, or authentication through smart cards. For example, a user with a Google account can use the Google as an Identity Provider to provide Identity Tokens, which can be used by the Identity Module to associate it with a Hyperty instance. Identity proxies are considered to act as intermediaries between the Identity Module and the specific Identity Provider Proxies (IdP Proxy), promoting a more flexible and adaptable solution. The IdP Proxy is the IDP ProtoStub that is responsible to handle the communication between the Identity Providers and the Identity Module.

The following figure illustrates this interaction:

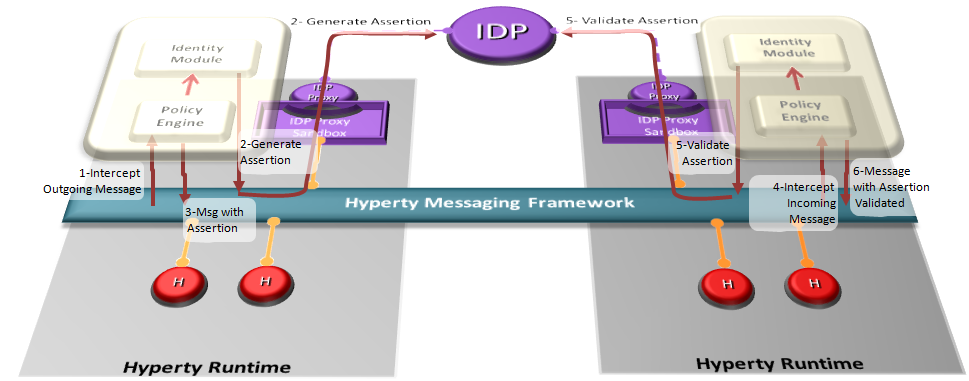


Figure - Interaction between the Identity Module and the Identity Provider

### Runtime Sandbox

The hyperty runtime implements sandboxing mechanisms that ensure the correct isolation of client JavaScript code (i.e., Hyperties, ProtoStubs, and Applications). Isolation means that client code is confined to execute within the address space of an independent sandbox. As a result, sandboxes prevent potentially malicious code from interfering with client code instances in co-located sandboxes or from accessing external resources in the surrounding environment (e.g., files, network, etc.). Communication outside the sandbox is possible through well defined channels. Both sandboxing mechanisms and communication channels implemented by the hyperty runtime are available to the application programmer throught specific APIs and are dependent on the targeted platform.

For the browser platform, sandboxing is enforced by leveraging native mechanisms provided by the browser API. The core runtime components execute inside an iFrame. The iFrame implements the core sandbox by isolating the code of the core runtime from the main window in which the application Javascript code is executed. Application code is therefore prevented from accessing directly to the memory address space of the core runtime. Communication between application and core runtime is possible only through a single and well defined entrypoint which allows them to exchange messages: method postMessage(). Hyperties and protoStub downloaded from different domains are executed in independent Web Workers. Web Workers effectively isolate their internal states from each other and from the core runtime. The postMessage() method constitutes the only communication bridge between the these components.

For the standalone platform, the sandboxing mechanisms we employ are similar to the browser platform. The main difference is that, instead of using a browser, we leverage Crosswalk to support standalone applications. Crosswalk is an HTML application runtime that allows us to execute the hyperty runtime as native mobile applications in Android and iOS devices without the need to install a full-blown browser. Mobile applications only need to be bundled with both Crosswalk webviews and the hyperty runtime code. Since a Crosswalk webview implements a Chromium-based runtime, it can seamlessly execute the hyperty runtime code that was implemented for the browser platform.

## Hyperty Development

This document provides guidelines for developers of Hyperties. It is recommended that you have already read:

* An Overview of the Hyperty Concept
* An Overview of the Hyperty Messaging Framework
* An overview on how Hyperties cooperate with each other through a Data Synchronisation model called Reporter - Observer
* An overview on the Hyperty Security and Trust Model

### Hyperty Concept

The Hyperty Concept is introduced [here](hyperty.md) as a secure user associated [microservice](http://martinfowler.com/articles/microservices.html), which can be deployed either on a web runtime environment, on an end-user device or on a networked server. The main characteristics of a Hyperty include:

* *Modular*: A Hyperty should be a self-contained module providing reusable service logic implementions in form of a script (e.g. a JavaScript file)
* *User association* : A Hyperty instance is associated to a “User” (e.g. Human beings, physical objects, physical spaces, organizations) through an Identity, even if this User can be anonymous in some cases.
* *Data Synchronization Communication*: Hyperties interact with each other through data synch objects by using the Reporter – Observer communication pattern.
* *Protocol Agnostic*: Through the protocol-on-the-fly concept, Hyperties are network protocol agnostic. In other words, the data synchronization communication between Hyperties is not dependent on a specific network protocol. Communication is accomplished via a common data schema that describes the data synch objects used.
* *GUI independent*: Hyperty should not provide Graphical User Interfaces. *to be clarified*
* *APIs*: A Hyperty can expose Javascript APIs within the runtime environment that can to be used by web applications

While designing and specifying service logics, it should be noted that Hyperties are not suitable for all use cases. In some case, making use of a simple reusable JavaScript file as library may suffice. The next section explains the criteria under which the decision to use a Hyperty or not could apply.

### Criteria to use the Hyperty Concept

These are guidelines to help developers decide if they should provide specific service logic as Hyperty or via a simple JavaScript library. Consider these as guidelines and not instructions. Before you embark on a new feature development, ask yourself the following questions:

* Is the feature delivery directly associated with a "User" (e.g. Human beings, physical objects, physical spaces, organizations)?
* Does the feature delivery involve communication between "Users"?
* Is the feature modular and reusable on different applications?
* Can the feature be delivered and developed by different stakeholders (i.e domain specific implementation)?

If the answers to the above questions are "YES" then most likely, you should go for the Hyperty Concept. Some examples are:

* Video Chat between human beings;
* a Human-being or a Data Analysis service collecting data from sensors in a Room.

The reTHINK Service Framework is what you want to look at next. The Service Framework provides APIs for developers to facilitate the development of Hyperties.

### Getting Started

So you have decided for the Hyperty Concept and now ask yourself where to start. This section describes the basic steps any developer needs to undertake to develop Hyperties.

1. Install the Service Framework

npm install git+git@github.com:reTHINK-project/dev-service-framework.git --save  
jspm install service-framework=github:reTHINK-project/dev-service-framework.git

1. Select or specify the descriptor of the Hyperty Data Objects in json-schema. Let's assume we define a simple Hello Hyperty object:

{  
 "$schema": "http://json-schema.org/draft-04/schema#",  
 "id": "HelloObject",  
 "type": "object",  
 "required": ["scheme","hello"],  
 "additionalProperties": false,  
 "properties": {  
 "scheme": {  
 "constant": "hello"  
 },  
 "hello": {  
 "type": "string"  
 }  
 }  
}

1. Then we can start developing the Hyperty Reporter of the previously defined Hello Hyperty Object. First the constructor where we are going to instantiate the Hyperty syncher. For that, we take as input parameter the Hyperty address (HypertyURL), the MiniBus used to receive and send data synchronization messages (bus), and any required configuration data.

constructor(hypertyURL, bus, configuration) {  
  
 let syncher = new Syncher(hypertyURL, bus, configuration);  
}

1. Then we prepare the creation of the Hyperty Hello Data Object by defining the Catalogue URL from where the previously defined Hyperty Data Object schema can be retrieved and initial data to be used in the creation:

\_this.\_objectDescriptorURL = 'hyperty-catalogue://example.com/.well-known/dataschemas/HelloWorldDataSchema';  
  
let hello = {  
 hello : "Hello World!!"  
};

1. The object is created and another Hyperty is invited to be an observer.

\_ syncher.create(\_this.\_objectDescriptorURL, [invitedHypertyURL], hello).then(function(dataObjectReporter) {  
  
 console.info('Hello Data Object was created: ', dataObjectReporter);

1. From the Observer side, in order to observe the Hello Data Object, it has to subscribe it, by passing its URL and the Catalogue URL from where its schema can be retrieved. As soon as the subscription is accepted the most updated version of the Hello Data Object is received ...

syncher.subscribe(\_this.\_objectDescriptorURL, ObjectUrl).then(function(dataObjectObserver) {}  
  
 console.info( dataObjectObserver);

1. Any change performed by the Reporter in the the object.

dataObjectReporter.data.hello = "Bye!!";

1. ... will be immediately received by the Observer:

dataObjectObserver.onChange('\*', function(event) {  
 // Hello World Object was changed  
 console.info(dataObjectObserver);  
 });

## Application Development

### Application vs Hyperty

A Hyperty is a module of software logic that is dynamically deployed in web runtime environments on end user devices, to execute session control and media flow management in a peer to peer manner. They are ready-to-use modules which are instantiated within the reTHINK runtime when required by the application. From the App developer perspective, Hyperties are similar to common Javascript libraries. Hyperty unique characteristics as described [here](hyperty.md) are transparent for the App developer. Through the Core Runtime, the required Hyperties and Protostubs are downloaded from the Catalogue server. This process is transparent for the developer of the final application, and of course, also for the final user of the application. If the Application requires some functionality or service provided by a Hyperty which has not been downloaded and instantiated yet, the runtime can get the code and instantiate it on the fly. The Application Developer only has to know in advance the Catalogue URL from where Hyperties are downloaded.

### How to use Hyperties

The usage of Hyperties is very simple and similar to the usage of any common Javascript library:

let runtime = new RuntimeLoader('hyperty-catalogue://example.com/.well-known/hyperty.runtime/default');

1. the App has to ensure the runtime is instantiated:

runtime.requireHyperty('hyperty-catalogue://example.com/.well-known/hyperty/HelloWorldReporter').then(  
  
 startUsingHpertyDeployed;  
  
 ).catch(function(reason) {  
 errorMessage(reason);  
});

1. then load and deploy the required Hyperty from the Catalogue

3- and invoke Hyperty functions from its API as a common Javascript Lib:

startUsingHpertyDeployed(hyperty){  
 hyperty.instance.hello();  
}

### How to adapt existing Applications

#### Standard web application

The diagram below shows how a standard application, which interacts with services provided by a service provider, currently looks like. There are several points which are not being considered such as Identity Management issues. If you need to interact with the service provided by CSP A, you need to use the library it provides, you need to authenticate to that specific service and you need to provide the logic in the Web Application to be able to produce and consume data from it.

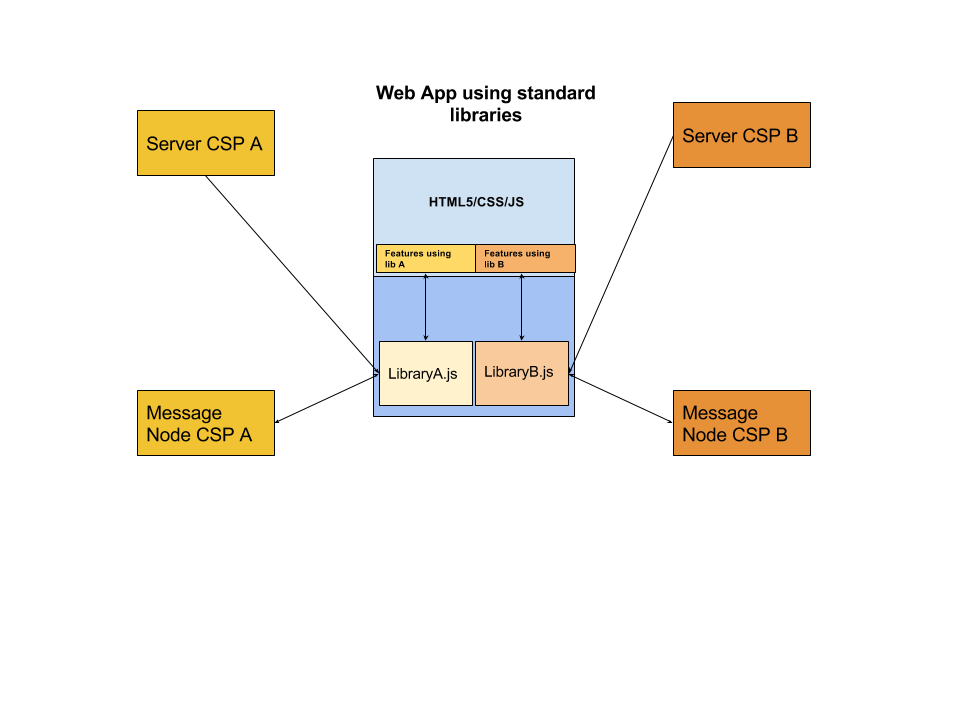


Figure - Standard App Diagram

If the Web App needs to interact with more than one CSP or service, typically it will need to download as many libraries as services and it may need to authenticate against the service provider with several identities and mechanisms. The code from different CSPs is running on the same sandbox so it can potentially interact with code from other libraries. This risk has been mitigated in the reTHINK architecture.

On the other side, we have only considered here Web Apps, but services provided by CSPs can be potentially used from a wide range of devices including constrainted devices for M2M application. That is the reason why the reTHINK client libraries has been to be executed also in M2M scenarios where no web broswers are involved.

#### reTHINK application

reTHINK web applications are similar to a common Web Application and have no impact at interface level, being impossible for an average user to distinguish between a standard application and a reTHINK application. However there are some settings available that gives the end user the power to take some decisions independently of the App consumed namely in terms of Identity to be used, Contact Lists and personal policies in general.

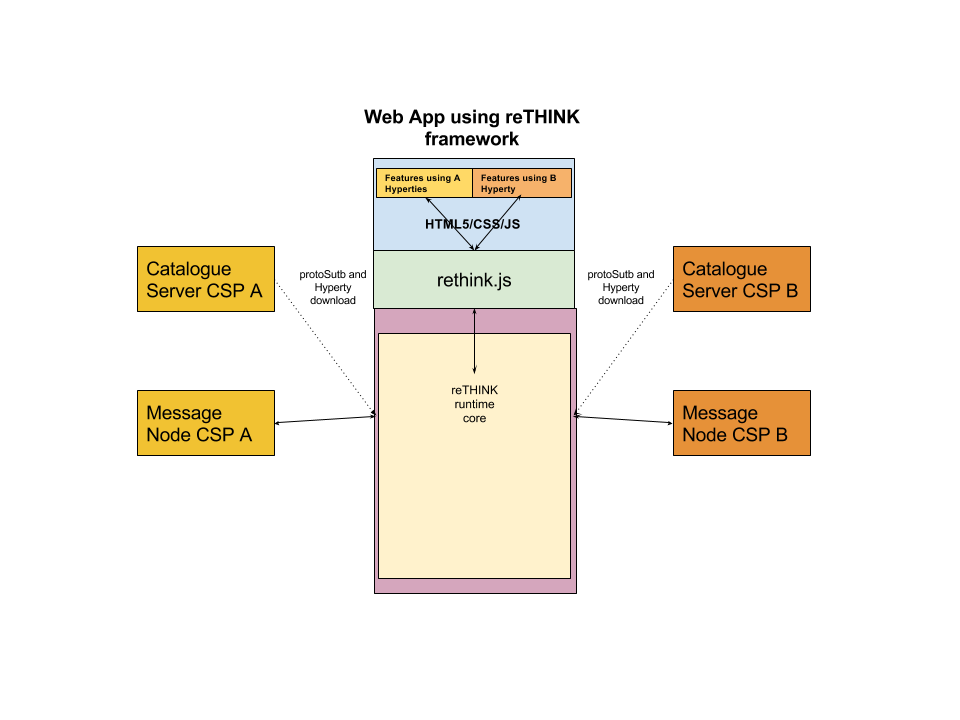


Figure - reTHINK App Diagram Black-box diagram

In the standard application, the developers need to know in advance the libraries and versions it has to download to build the aplication. In reTHINK the functionality is provided by hyperties which are dynamically loaded in the runtime on-demand. The interaction with the Hyperties which can be executed in different sandboxes is done through commands and events from the Web application.

The diagram below depicts a higher level picture of the Core Runtime architecture which is transparent to App developers. This diagram has been created for didatic purposes and it does not include all the elements. For example it does not contain the modules in charge of dealing with identity management.

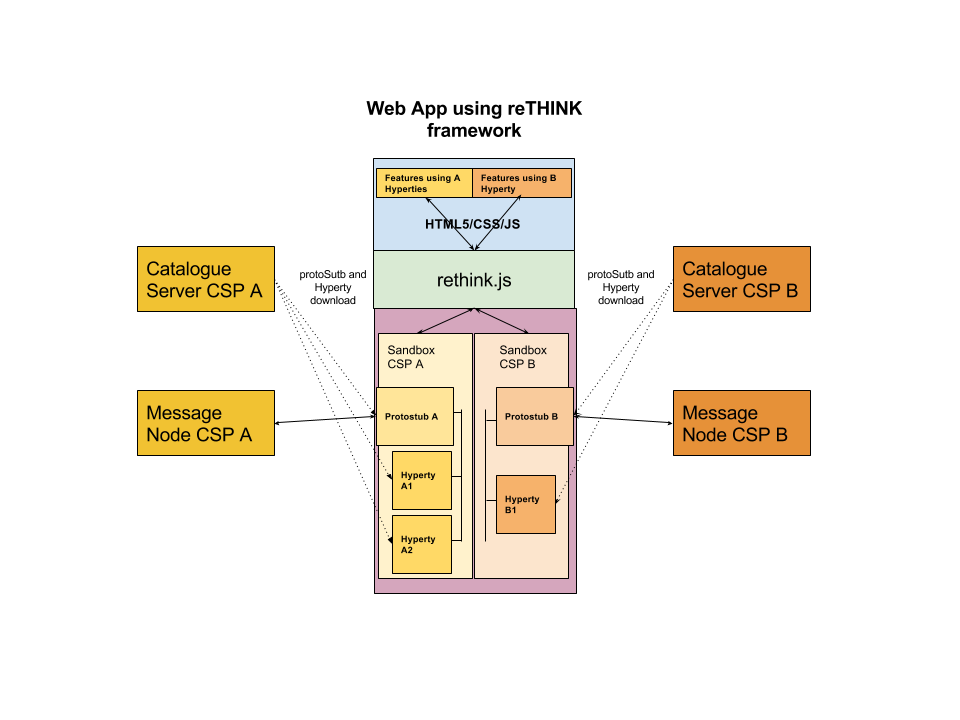


Figure - reTHINK App Diagram

##### Security: sandboxes

The Web developer does not have to deal with low level details of the architecture. The sandboxes and the management of hyperties and protostubs is done by the core framework. The sandboxes allow to isolate code from different providers reducing the risk of suffering cross-site scripting attacks. There are special cases where the Hyperty needs to be executed in the same sandbox as the main web application, for example, when the Hyperty needs to access the WebRTC API.

##### Compatibility: Protostubs

In a standard Web Application, the developers needs to know in advance with which services providers it will be necessary to interact. The number of protocols an application can speak is limited in implementation time and it can not change without modifying the code. In reTHINK the protocol-on-the-fly concept is used. If you need to interact with a service which uses protocol A, the framework will provide you on-the-fly a piece of code called protostub which will be executed in the right sandbox. This protostub will speak protocol A and it wil expose a common API to the Hyperty Core Runtime. The Web Developer will not need to deal with this complexity.

##### Identity management.

The identity management is normally coupled to the service logic and there are many different standard protocols for authentication and Identity management that makes it harder to achieve interoperability between different services. The reTHINK framework decouples the service logic from the Identity Management logic and provides a common Identity Managemet API that is agnostic of protocols used. The protocol-on-the-fly mechanism is also used to provide you on-the-fly the right protostub (here called IDP Proxy) to interact with each Identity Provider selected by the end-user.

## Message Node and Protostubs Development

### Overview

The protocol stubs (AKA protostub) play a central role in the protocol on-the-fly concept. A stub is the piece of code that a reTHINK runtime downloads, instantiates and executes on-the-fly in order to exchange messages with a backend system from a foreign or even from the own domain. From the runtime's point of view the stub is the required "glue" between the reTHINK Message Model and the backend domain's protocols. The stub implements a well defined interface for the bi-directional asynchronous exchange of messages and hides all potential complexity of protocol translations for the interoperability with the backend domain.

The communication endpoint of a stub in a domains backend is the Messaging Node (MN). The MN and the stub build a unit that shall be designed and implemented together. The implementor of a protocol stub and the corresponding MN has to take some decisions:

1. is the Message Node able to route native reTHINK JSON Messages and no protocol translation is required in the protostub?
2. is it possible to add with a reasonable effort, Message address allocation and Subscription Management support services functionalities as well as required connectors to interact with reTHINK back end services (including Domain Registry, Global Registry and Identity Management support services) to the Message Node?

If answers to above questions are yes, probably the most appropriate Message Node topology is the standalone message node model where all functionalities are provided by a single message node service and its associated protostub. This topology is used by [Vertx Message Node](https://github.com/reTHINK-project/dev-msg-node-vertx) and [NodeJS Message Node](https://github.com/reTHINK-project/dev-msg-node-nodejs).

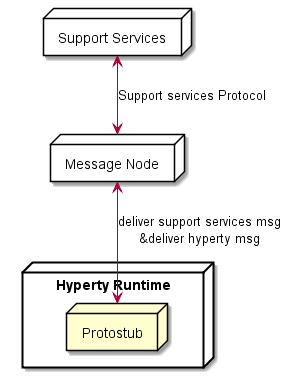


Figure - Message Node Standalone Topology

If answer to question 1 is negative, the potential complexity to be placed in the stub itself should be evaluated: Shall the stub do everything that is necessary to translate the protocol to the backend domains specifics? Or shall the stub just forward messages and let the MN perform the major tasks of the protocol translations? These are some hints that the developer should take into account:

1. Does the stub have dependencies to additional libraries? This might blow up the size of the stub and may complicate its deployment. Perhaps there is a chance to avoid some external dependencies?
2. Do any parts of the stub and it's dependencies underlie special restricting licenses or do they contain code that holds intellectual properties that shall be protected? Since the code is downloaded to an unknown, "strange" environment this might be an issue.
3. How many resources (network, processing, memory etc.) does the stub require? Are these requirements compatible with all addressed runtime platforms?

These questions shall be kept in mind, when the design decisions for a stub/MN couple are made. If one of the above questions can be answered with yes, then it might perhaps be an option to implement a basic stub that uses a simple connection mechanism like a WebSocket or similar to forward the reTHINK messages directly to the MN. In this case the MN itself would be responsible for the required protocol translations on the server side for its domain.

An example for such a situation is the [Matrix.org based MN](https://github.com/reTHINK-project/dev-msg-node-matrix) and its stub. The decision was made to let the stub just forward reTHINK messages and therefore keep it simple and small. The implementation of the Matrix.org client logic was done on the MN side. If the stub had implemented a full Matrix.org client, there would have been a set of dependent SDK-libraries with their own set of dependencies each. Furthermore a Matrix.org client produces additional overhead traffic that should be restricted to the MN internal system and therefore be kept away from the runtime device.

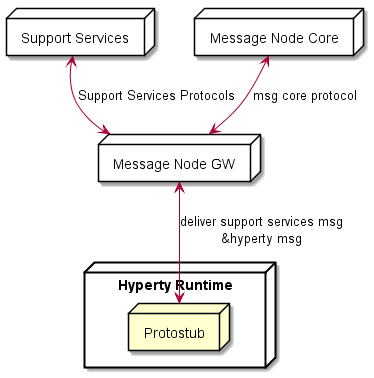


Figure - Message Node Standalone Topology

Another aspect to be taken into account is whether the Message Node is based on an existing Messaging solution that is already in production (e.g. core IMS, cloud messaging like pubnub, firebase, etc). In this case Messaging Node specific functionalities (allocation manager, subscription manager, registry connector) can be provided by a separated Support Service server, while Hyperty messages are delivered to the messaging core. This means, the protostub would handle two different protocols connections as shown below.

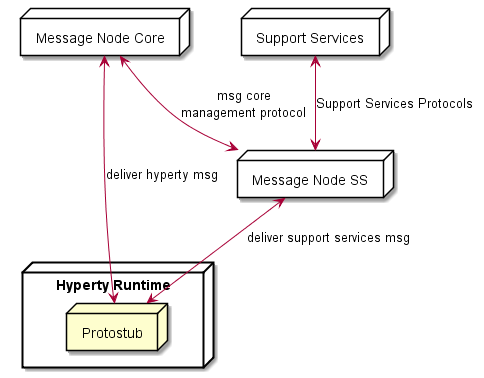


Figure - Message Node with separated Support Services

### Messaging Model

#### General message format

A reTHINK message is a standard JSON Object with a fixed set of header fields and a variable message body. These are the common header fields:

|  |  |  |
| --- | --- | --- |
| **name** | **type** | **description** |
| id | numeric | an identifier used to associate RESPONSE messages to the initial REQUEST message. It should be noted that the REQUEST.id MUST be incremented every time a new REQUEST message is created. |
| from | [URL](https://github.com/reTHINK-project/dev-service-framework/blob/develop/docs/datamodel/address/readme.md) | URL of a Hyperty instance or user associated with it |
| to | [URL](https://github.com/reTHINK-project/dev-service-framework/blob/develop/docs/datamodel/address/readme.md) | One or more [URLs](https://github.com/reTHINK-project/dev-service-framework/blob/develop/docs/datamodel/address/readme.md) of Message recipients. According to the URL scheme it may be treated in different ways by the MN. |
| body | JSON-Object | The message body according to the type that is identified by the type attribute in the message header. |

#### Message Body

The Message Body is a JSON object that varies according to the message type, specified in the message header. Currently following types of Message bodies are specified in the reTHINK specification:

* CreateMessageBody
* ReadMessageBody
* UpdateMessageBody
* DeleteMessageBody
* ForwardMessageBody
* ResponseMessageBody
* SubscribeMessageBody
* UnsubscribeMessageBody
* ExecuteMessageBody

Message bodies can contain JWT tokens for Access Control or for Identity Assertion purposes. See [here](../specs/dynamic-view/identity-management/user-identity-assertion.md) for more details.

The "MessageBody.via" attribute contains a list of all Protostub addresses (Protostub) that the message has been passed through. It is used to prevent infinite cycles in the Hyperty Messaging Framework.

Detailed specifications of these Message bodies can be found at [Message Model](../datamodel/message/readme.md).

#### Request - Response transactions

A Response to a Request message should follow this rule:

Response.from = Request.to  
Response.to = Request.from  
Response.id = Request.id

The Request.id MUST be incremented every time a new Request message is created.

### APIs

#### The ProtoStub API

#### The ProtoStub API

The interface that a protocol stub has to implement is kept very small and simple by intent. describe main flow as UML for: - stub connection - message routing

A protocolStub is constructed with a set of parameters that ensures that the stub can be uniquely identified, can connect to its backend Messaging Node and can communicate with the messaging bus in the runtime.

new ProtoStub(runtimeProtoStubURL, busPostMessage, configuration)

*Parameters:*

|  |  |  |
| --- | --- | --- |
| **name** | **type** | **description** |
| runtimeProtoStubURL | URL.RuntimeURL | A URL allocated by the runtime that uniquely identifies this protocolStub. |
| busPostMessage | Message.Message | The runtime BUS postMessage function to be invoked on messages received by the protocol stub. |
| configuration | ProtoStubDescriptor.ConfigurationDataList | Configuration data that is retrieved from the protocolStub descriptor. This data is implementation-specific and ensures that the Stub can address and connect its own Messaging Node. |

##### Methods

The connect method establishes the connection between the protocol stub and the backend messaging node.

connect(identity)

**Note:** The "connect" method will not be directly invoked by the runtime implementation. Rather it is expected that the stub maintains its connection state internally. Whenever the runtime intents to send a message via the postMessage method, the stub shall auto-connect to the Messaging Node and attempt to keep this connection open until it explicitly receives a "disconnect" invocation.

*Parameters:*

|  |  |  |
| --- | --- | --- |
| **name** | **type** | **description** |
| identity | IDToken | An optional identity token that can be used to authenticate this stub connection against the backend messaging node |

The disconnect method is used to explicitly disconnect a stub from its messaging node. Such a disconnect can be used to release and clean up resources in the stub and also on the backend side in the messaging node.

disconnect()

The postMessage method is used by the runtime to send messages through the protocol stub to connected backend server.postMessage(message)

|  |  |  |
| --- | --- | --- |
| **name** | **type** | **description** |
| message | Message.Message | The message to be dispatched by the protocol stub. |

##### Events

A protocol stub emits events to communicate its own connection state to the runtime. Whenever the stub gets connected or disconnected, it uses the "busPostMessage" to send a message to the runtimes message bus. These Event messages are defined as follows:

{  
 "type": 'update',  
 "from": runtimeProtoStubURL,  
 "to": runtimeProtoStubURL + '/status',  
 "body": {  
 "value": "connected|disconnected"  
}

The runtimeProtoStubURL is the URL that was provided as first parameter of the Stub constructor. The value in the message body is either "connected" or "disconnected".

### Message Node functionalities and main procedures

#### Stub identification and resource management

The MN is the connectivity endpoint for stubs that are deployed in several runtimes. From the viewpoint of the MN, each stub represents one runtime. It is the task of the MN to identify a stub connection, and to manage the life-cycle of the assigned server side resources. The actual "handshake mechanisms" between the stub and the MN are left implementation specific.

A valid method for the MN to identify a stub connection is to use the "runtimeURL", which each stub is constructed with in the runtime. If the stub provides this url during the connection handshake procedure, then the MN can identify the stub/runtime, even after a potential re-connect, e.g. due to temporary loss of network connectivity.

It is the responsibility of the MN to release resources if the "disconnect" method was invoked on the stub . This is the official indication that the runtime does not need this stub connection anymore and it has released the stub. In the alternative case, that a stub was not sending messages for a longer period, but was also not officially disconnected, it is up to the MN implementation to run a kind of garbage collection mechanism to release stale resources.

#### Address Allocation

As soon as an entity in a runtime wants to be accessible from another runtime, this entity must be addressable. Since a MN is the central message routing point for a domain it is the MNs task to create these addresses and to assign them to the requesting runtime. The resulting internal allocation table stores the relation of the allocated addresses to the stub connections and enables a proper routing of messages between the runtimes.

The MN must support address allocation for Hyperties as well as for data object. The general format of an allocation message is as follows:

"id" : "<1>"  
"type" : "CREATE",  
"from" : "hyperty-runtime://<sp-domain>/<runtime-instance-identifier>/registry/allocation",  
"to" : "domain://msg-node.<sp-domain>/<type>-address-allocation",  
"body" : { "scheme" : <scheme>, "childrenResources" : [{"<resource-children-name>"}], "value" : {"number" : <integer> , "allocationKey" : "<key>"} }

where the "number" attribute stands for the number of requested addresses, the "scheme" defines the requested url scheme (or protocol) of the address and the "allocationKey" serves as identifier of this set of allocated addresses. This key can be used to identify addresses to be deleted later on.  
The MN must intercept such messages and respond with a message like:

"id" : "<1>"  
"type" : "RESPONSE",  
"from" : "domain://msg-node.<sp-domain>/<type>-address-allocation",  
"to" : "hyperty-runtime://<sp-domain>/<runtime-instance-identifier>/registry/allocation",  
"body" : { "code": 200, "value" : {"allocated": ["<scheme>://<sp-domain>/<identifier>", ...]} }

The format of the generated part of the url is implementation specific.

The MN must de-allocate addresses, if it receives a DELETE message of this format:

"id" : "<3>"  
"type" : "DELETE",  
"from" : "hyperty-runtime://<sp-domain>/<runtime-instance-identifier>/registry/allocation",  
"to" : "domain://msg-node.<sp-domain>/<type>-address-allocation",  
"body" : { "resource" : "<key>" }

The "key" value in the body serves as an identifier of the previously allocated address(es).

For more detailed information about the allocation Messages refer to [Address allocation messages](../specs/messages/address-allocation-messages.md).

#### Interaction with the Domain Registry

The allocation of a unique address is only the first step on the way to make an entity (hyperty or data object) discoverable and usable from another runtime. In order to make it discoverable the allocated addresses must be registered in the domain registry component. The interaction with the domain registry is also the task of the MN. The MN just has to route messages from a runtime that address the subdomain of the MNs own url towards the Domain Registry connector and a callback is used to deliver responses back to the runtime. In this way, the MN is not aware of the Registration messages payload. The full specification of these messages can be found here [Registration Messages](https://github.com/reTHINK-project/dev-service-framework/blob/d3.2-working-docs/docs/specs/messages/registration-messages.md)

#### Subscription management

A core concept in the reTHINK architecture is that Hyperties interact with each other by exchanging and synchronizing their managed data objects based on the [Reporter - Observer pattern](p2p-data-sync.md). The MN supports this concept by allowing observers (Hyperties, running in one or more runtimes) to subscribe for changes of certain allocated data object urls deployed in other runtimes. Whenever a Hyperty runtime reports a change in a monitored data object it sends a change message to the MN. The "to" address of this message will just be the allocated address of the updated data object, not the address of the subscribers directly.

In order to route such object change messages to the subscribed listeners, the MN has to maintain an own list of subscribers per allocated data object. Therefore the MN must intercept subscription messages which have the following format:

"id" : "1"   
"type" : "SUBSCRIBE",  
 "from" : "hyperty-runtime://<observer-sp-domain>/<hyperty-observer-runtime-instance-identifier>/sm",  
 "to" : "domain://msg-node.<observer-sp-domain>/sm",  
 "body" : { "resource" : "<ObjectURL>" , "childrenResources" : [{"<resource-children-name>"}]}

This message of type "SUBSCRIBE" is addressed to "domain://msg-node./sm", which is the identifier of the MNs "Synch Manager (sm)" component. In the body the most important field is the "resource", which contains the allocated address of the object that shall be subscribed by the runtimes sync manager (as identified by the "from" field).

The MN must extract the from the body and assign this URL internally to the given HypertyURL contained in the "from" URL. This means for the MN that every future "changes"-message to this ObjectURL must be forwarded to the Hyperty Runtime Instance. If the "childrenResources" arrays contains values, than additional assignments must be created for each + / + .

After extraction of the parameters and the creation of the assignments, the MN must respond with a message of code 200 back to the runtime.

"id" : "1",  
 "type" : "RESPONSE",  
 "from" : "domain://msg-node.<observer-sp-domain>/sm",  
 "to" : "hyperty-runtime://<observer-sp-domain>/<hyperty-observer-runtime-instance-identifier>/sm",  
 "body" : { "code" : "2XX" }

*NOTE:* The procedure to un-subscribe from data object changes looks very similar to the above described subscribe procedure. The message to intercept is then of type "UNSUBSCRIBE". The MN has to remove the previously mapped assignments and respond back with a code 200 message.

If the MN later on receives a message from a reporting Hyperty that its data model has changed this message will look like this:

"id" : "3",  
 "type" : "UPDATE",  
 "from" : "<ObjectURL>",  
 "to" : "<ObjectURL>/changes",  
 "body" : { "value" : "changed value" }

Note that the "from" and "to" fields just contain the and the "to"-field has the suffix "/changes". When the MN receives such a message, it must look up for all subscribed listeners to this and forward the message to them.

A more detailed specification can be found at [Data sync messages](https://github.com/reTHINK-project/dev-service-framework/blob/d3.2-working-docs/docs/specs/messages/data-sync-messages.md).

#### Policy decisions and enforcement

Message nodes are responsible for the interaction of runtimes in their own domain with runtimes from foreign domains by offering protocol stubs to these external runtimes. However the operators of a certain domain need a mechanism to control these domain interactions and to potentially block or limit certain combinations of message exchange.

In order to achieve this, a MN must provide a hook in the message flow that allows to apply policy based decisions to the routing. These policy must be manageable by the domain Policy Manager.

#### Protocol on-the-fly engine

The basic operation mode of a MN is that it is connected by runtimes directly via the provided protocol stubs. A message received from one runtime will be forwarded to another runtime which must also be directly connected through a stub. This is a classic "triangular" messaging architecture. The triangular message flow looks like this:

RuntimeA --> StubB --> MN-B --> RuntimeB

For future iterations of the reTHINK messaging it is intended that the MNs also support a "trapezoid" architecture for inter-domain communication. In contrast to the triangle, each runtime will only have a connection with the MN from its own domain. If one runtime wants to send a message to another one from another domain, it will not be runtime itself that downloads and instantiates the stub of the foreign domain. It would be the domains MN instead that has to do this.

The trapezoid message flow will then look like this:

RuntimeA --> StubA --> MN-A --> StubB --> MN-B --> RuntimeB

and vice versa. This implies that in future versions the MN has to implement a module for the proper downloading, instantiation and operation of foreign stub in a sandboxed environment, just like the runtimes are already doing it.

### Message routing procedure

This section tries to summarize all the descriptions of the individual MN components from above and describe the basic messaging handling and routing procedures inside a MN. It uses a pseudo-code like format to describe the order of the operational steps.

Several checks must be applied:

* Is it a routable reTHINK message?
  + i.e. does it contain a "from" and "to" field?
  + if not --> reject / ignore
  + stop
* To be confirmed: Identity-token verification ?
* Is it an allocation management message?
  + allocate / de-allocate addresses
  + return proper RESPONSE message
  + stop
* Is it a registration management message?
  + perform the requested (asynchronous) interaction with the domain registry
  + return the result of this interaction in a proper RESPONSE message
  + stop
* Is it a subscription management message?
  + extract the DataObjectURL and potential childrenResources from the message body
  + perform the requested assignments / de-assignments to the internal subscriber mappings
  + stop
* Is the message type == UPDATE and the "from" address one of the previously subscribed DataObjectURLs?
  + Is the "to" address == "from" + "/changes"?
    - retrieve the corresponding runtime URL from the subscriber mappings
    - forward this message to the retrieved runtime URL via the proper stub
    - stop
* This seems to be a "normal" message.
  + extract the "from" address and remember its relation to the stub that has sent this message
  + (this is required to find the correct return path for a subsequent response to this address)
  + extract the "to" address and investigate the corresponding stub
  + if the "to" address corresponds to a connected stub ()
    - forward the message through this stub
  + else
    - (the "to" address points to a domain that is not currently connected via a stub)
    - discover, download, instantiate and use a stub to this foreign domain (trapezoid architecture)
  + stop

### Protostub Source Code Examples

#### Stub construction and activation

Stubs are provided by different vendors and developers and of course they have different naming conventions. In order to provide a common instantiation scheme a convention was defined additionally to the interface that ProtoStubs have to implement. The convention is that each stub modules must export a default activation function that is used by the runtimes to obtain a stub instance with a given set of parameters.

export default function activate(url, bus, config) {  
 return {  
 name: 'MatrixProtoStub',  
 instance: new MatrixProtoStub(url, bus, config)  
 };  
}

This activation function hides the internal naming and just returns an object that provides an implementation of the methods defined in the ProtoStub interface. The parameters of this function correspond directly to the previously described parameters of the Stub constructor.

#### Auto connect mechanism

As mentioned as a side note in the API description of the ProtoStub's connect method, the stubs are expected to support an auto connect mechanism. This is because the runtime will not explicitely invoke the connect method itself. Instead it just sends messages via the messaging bus to the stub and assumes that the stub takes care of its own connection state.

A simple approach to implement this behavior in the stub is to maintain a flag that indicates whether the connection to the MN shall be kept open or not. This flag could be set to TRUE, as soon as the first message is being sent and to FALSE if the stub receives a "disconnect" command from the runtime. If for instance a network problem causes an interruption of the connection between stub and MN, the stub would attempt to re-connect as soon as the next message shall be sent.

This is, how the method to send a message could look like:

\_sendWSMsg(msg) {  
 if ( this.\_assumeOpen )  
 this.connect().then( () => {  
 this.\_ws.send(JSON.stringify(msg));  
 });  
}

If there is an explicit invocation of the "disconnect" method of the stub the stub will close the connection to the MN and set the keep alive flag off.

disconnect() {  
 this.\_ws.close();  
 this.\_assumeOpen = false;  
}

#### Connection events

The stub must emit a "connect" or "disconnect" message to the bus whenever its connection state changes. The following method can be used to encapsulate this:

\_sendStatus(value, reason) { let msg = { type: 'update', from: this.\_runtimeProtoStubURL, to: this.\_runtimeProtoStubURL + '/status', body: { value: value } }; if (reason) { msg.body.desc = reason; }  
  
this.\_bus.postMessage(msg); }

The expected "value" parameter is either "connected" or "disconnected". Optionally a reason can be specified that will be placed int the body of the message.

If the connection to the MN is established via a Websocket, then the sending of the corresponding event messages can be triggered in the "open" and "close" handlers of the Websocket.

\_onWSOpen() { this.\_sendStatus("connected"); }  
  
\_onWSClose() { this.\_sendStatus("disconnected"); }

#### Integration with the Messaging Bus of the Runtime

Protocol stubs are tightly integrated with the messaging bus of the runtime. This integration is bi-directional. A reference to the messaging bus is provided as second paramenter of the stub constructor.

In order to receive messages from the runtime's messaging bus, the stub has to add itself as a listener. This can be done directly in the stubs constructor by adding such a code snippet:

this.\_bus.addListener('\*', (msg) => {  
 this.\_assumeOpen = true;  
 this.\_sendWSMsg(msg);  
});

Whenever now the stub receives a message via this listener callback it sends it forward (in this case via a Websocke connection) to its MN.

For every message that is received from the MN, the stub forwards this message to the bus by using its postMessage method like shown here:

// parse msg and forward it locally to the runtimes messaging bus  
\_onWSMessage(msg) {  
 this.\_bus.postMessage(JSON.parse(msg.data));  
}

# Specification changes

This chapter provides a short summary of major changes done in the specification of the Runtime and Message Node, according to Phase 2 implementation feedback.

## Hyperty Runtime

The Hyperty Runtime supports the execution of Hyperties. The Hyperty Runtime architecture follows a security by design approach since it was highly influenced by a careful security analysis. Different types of components with different origins are deployed and executed in isolated sandboxes. As highlighted in Figure 19, comparing with the original specification, two new components were added to the Hyperty Core Runtime:

The **Persistence Manager** was added to provide a transversal data storage functionality to be used by any Core Runtime Components.

The **Sync Manager** was addedin in order to support Hyperty Data Objects synchronisation by handling creation and subscriptions requests.

In addition, the **CatalogueProtostub** from the **Runtime User Agent** was renamed to **Runtime Catalogue** and has a few more functionalities. It manages the descriptors of deployable components and Hyperty Data Object schemas that are downloaded from the Service Provider Catalogue via the [Catalogue Service interface](https://github.com/reTHINK-project/architecture/blob/master/docs/interface-design/Interface-Design.md#73-catalogue-interface). The Runtime Catalogue ensures synchronisation with Back-end Catalogue servers.

The **QoS User Agent** that manages network QoS was not implemented in Phase 2.

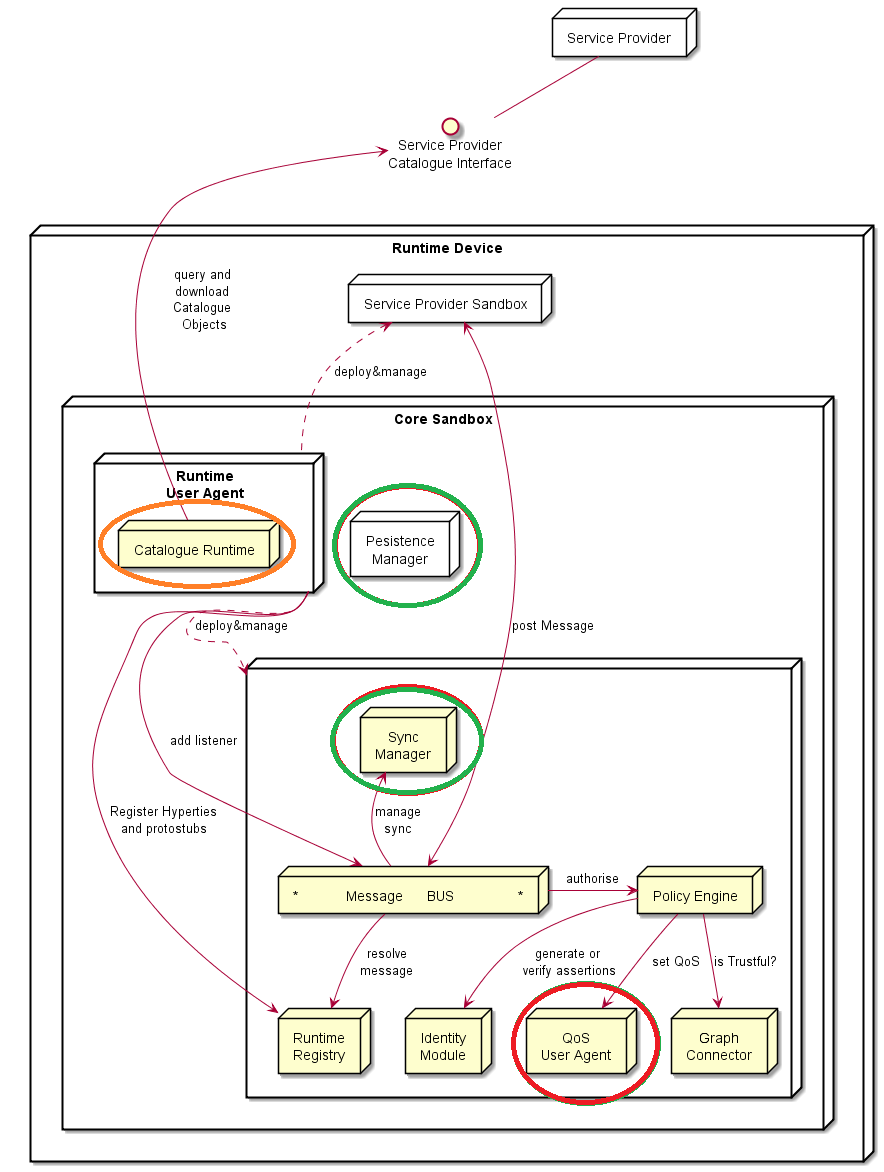


Figure – Changes in the Hyperty Runtime Core

## Messaging Node

The Messaging Node main functionality is to route Hyperty Messages among Hyperty Runtimes. As highlighted in Figure 20, comparing with the original specification, only one new component was added to the Messaging Node Architecture:

The **Subscription Manager** was added to support the new Runtime Sync Managers. It is in charge of handling Subscription and Unsubscription requests from Runtime Sync Managers in order to manage the Data Sync Stream routing path in the Message Node.

In addition, there is only the renaming of the “Access Control PEP” component to “Policy Engine” without any change on its functionalities.

The support of Protostubs in the Messaging Node will be added in phase 2.

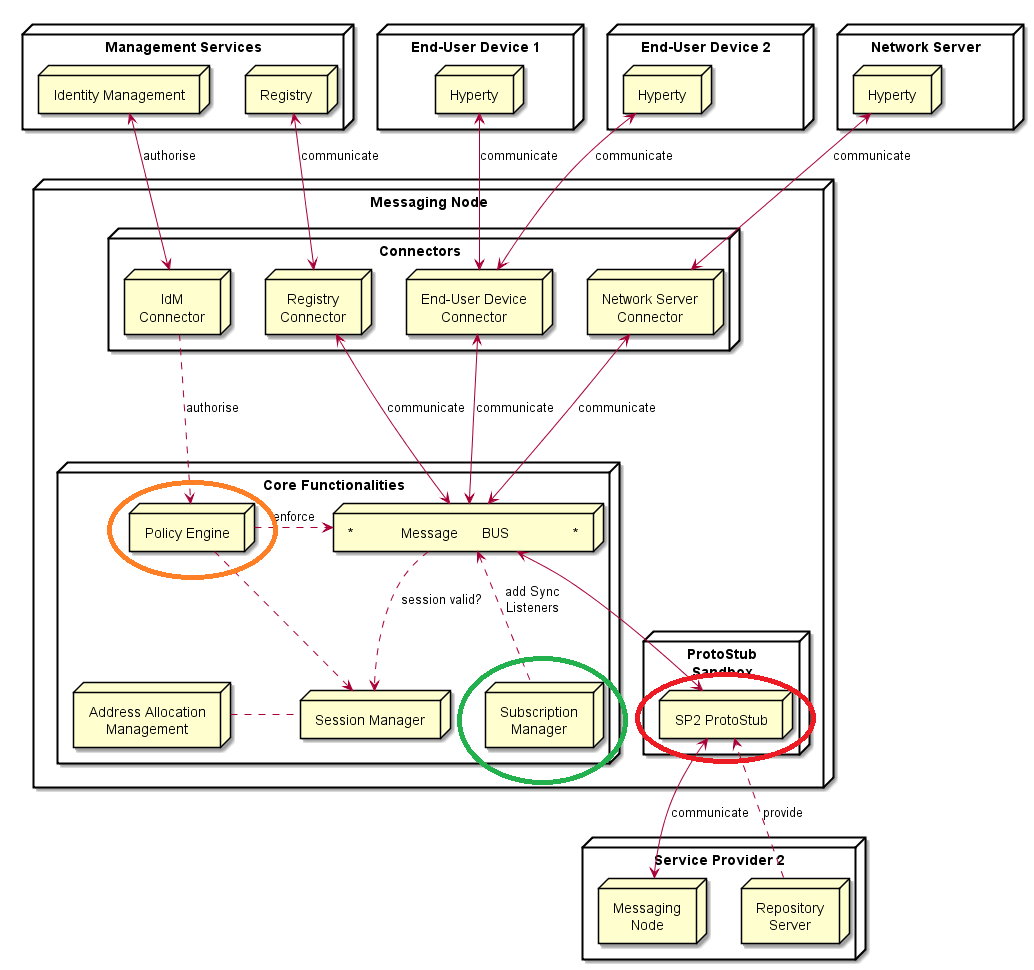


Figure – Changes in the Messaging Node Architecture

# References

1. https://github.com/reTHINK-project
2. Deliverable D2.1, “Framework Architecture Definition”, 31-07-2015.
3. Deliverable D2.2, “Data Models and Interface Specification of the Framework ”, 30-08-2015
4. Deliverable D3.1, “Hyperty-Runtime-and-Hyperty-Messaging-Node-Specification”, 30-09-2015.
5. Deliverable D4.1, “Management and Security features specifications”, 30-09-2015.
6. Deliverable D4.2, “Implementation of Governance and identity management components for Phase 2”, 29-02-2016.
7. <https://github.com/reTHINK-project/dev-runtime-core>
8. <https://github.com/reTHINK-project/dev-runtime-browser>
9. <https://github.com/reTHINK-project/dev-runtime-nodejs>
10. <https://github.com/reTHINK-project/dev-msg-node-vertx>
11. <https://github.com/reTHINK-project/dev-msg-node-matrix>
12. <https://github.com/reTHINK-project/dev-msg-node-nodejs>
13. <https://github.com/reTHINK-project/dev-service-framework>
14. Paulo Chainho, Kay Haensge, Steffen Druesedow, Michael Maruscheke, “Signalling-On-the-fly: SigOfly, WebRTC Interoperability testbed in contradictive Deployement Scenarios”, Proc. 18th Int’l Conf. Intelligence in Next Generation Networks (ICIN), 2015.
15. <https://github.com/reTHINK-project/dev-catalogue>
16. <https://github.com/reTHINK-project/dev-registry-domain>

1. <http://martinfowler.com/articles/microservices.html>
2. <https://en.wikipedia.org/wiki/Edge_computing>
3. https://en.wikipedia.org/wiki/Fog\_computing
4. https://developers.google.com/v8/
5. https://nodesecurity.io/resources
6. Components Documentation

## Hyperty Runtime Core

* [Overview](#overview)
* [User View: How to include the Hyperty Runtime Core in other Projects](#user-view)
* [Developer View](#developer-view)
* [Example](#example)
* [Tasks](#tasks)
* [Notes](#notes)

### Overview

This repository contains the source code and associated documentation of the core components required to support the deployment and execution of Hyperties in user devices or in network servers. More information about the Hyperty concept and the reTHINK framework in general is provided [here](https://github.com/reTHINK-project/dev-service-framework/blob/master/README.md).

The Hyperty Runtime architecture follows a security by design approach since it was highly influenced by a careful security analysis where different types of components are executed in isolated sandboxes. Thus, components downloaded from a specific Service Provider are executed in sandboxes that are different from the sandboxes used to execute components downloaded from another service provider. Communication between components running in different sandboxes is only possible through messages exchanged through a Message Bus functionality provided by the Hyperty Runtime Core Sandbox. On the other hand, and according to the [ProtoOFly concept](https://github.com/reTHINK-project/dev-service-framework/blob/master/docs/manuals/hyperty-messaging-framework.md#protocol-on-the-fly-protofly-and-protostubs), the protocol stub is executed in isolated sandbox and provides the bridge for the Hperty Runtime to communicate with associated Service Provider. The detailed specification of the Hyperty Runtime Core is provided [here](docs/specs/readme.md).

Hyperty Core Runtime components are platform agnostic and are to be included in platform specific Hyperty Runtimes, like Web Browsers and Nodejs based platforms.

### User View

**How to include the Hyperty Runtime Core in other Projects**

How to include this repository in other runtime platforms, like [dev-runtime-browser](https://github.com/reTHINK-project/dev-runtime-browser) or [dev-runtime-node](https://github.com/reTHINK-project/dev-runtime-node);

#### [Browser Runtime](https://github.com/reTHINK-project/dev-runtime-browser)

Verify these use cases:

1. if you will create a new repository, you can use this template, and can configure your development environment;
2. if you already have an respository cloned;

for both cases you just have to run the command:

jspm install runtime-core=github:rethink-project/dev-runtime-core@dev-0.2

and on javascript code you just need to import the script like other modules;

import RuntimeUA from 'runtime-core/dist/runtimeUA';  
import {Sandbox, SandboxRegistry} from 'runtime-core/dist/sandbox'  
import MiniBus from 'runtime-core/dist/minibus';  
  
console.log('Runtime: ', RuntimeUA);  
console.log('Sandbox: ', Sandbox, SandboxRegistry);  
console.log('MiniBus: ', MiniBus);

#### [Nodejs Runtime](https://github.com/reTHINK-project/dev-runtime-node)

[dev-runtime-node

npm install github:rethink-project/dev-runtime-core#dev-0.2 --save

after this you can require the runtime-core like other modules on node;

var RuntimeUA = require('runtime-core').runtimeUA;  
  
var runtime = new RuntimeUA();

if you found some issues, please submit them into the respective repository;

### Developer view

#### Setup Environment

On the first time you are cloning this repository, you need to run the command:npm run init-setup

After running successfully this command you will have 2 folders (node\_modules and vendor), these folders are excluded from the commit process, and are only for development.

if you already have the project configured on your machine, you only need run the next command to add new dependencies:npm install jspm install

**Private Repository Note**

if you have problems with the npm install command, like “access was forbidden”, “404 not found”, and have the service framework module reference, it is an authentication problem;

you may need following the steps present on [Github Help](https://help.github.com/articles/generating-ssh-keys/). and select operation system you are using.

This could happen because it is a private module and need your GitHub authentication to allow cloning the repository.

If you have some troubles with authentication on windows using the Git Shell, you can try [caching your GitHub password](https://help.github.com/articles/caching-your-github-password-in-git/#platform-windows). This should avoid the constant prompt for username and password;

**Instalation through jspm**

We need configure jspm config using github tokens, for that, following this (based on issue [3](https://github.com/reTHINK-project/dev-runtime-browser/issues/3)):

1. [Here](https://github.com/settings/tokens), generate token with public\_repo permission enabled
2. Save the token generated;
3. Execute the command jspm registry config github and you’ll be asked for the credentials;
4. Now you can execute command jspm install -y and the runtime-core or jspm install runtime-core=github:reTHINK-project/dev-runtime-core or only jspm install;

**Issues**

if you have some trouble with the environment, you can open an issue;

#### Javascript Environment

JavaScript code should be written in ES6. There are direct dependencies from nodejs and npm, these can be installed separately or in conjunction with [nvm](https://github.com/creationix/nvm)

##### Dependencies

* nodejs
* npm
* karma - Make the communication between unit test tool and jenkins. See more on [karma](http://karma-runner.github.io/0.13/index.html)
* mocha - Unit test tool. See more on [http://mochajs.org](http://mochajs.org/)
* jspm - Don’t need compile the code, it uses babel (or traucer or typescript) to run ES6 code on browser. Know more in [jspm.io](http://jspm.io/)
* gulp - Automate and enhance your workflow. See more about gulp on [gulp](http://gulpjs.com/)

##### Code Style and Hinting

On the root directory you will find **.jshintrc** and **.jscsrc**, these files are helpers to maintain syntax consistency, it signals syntax mistakes and makes the code equal for all developers.

* [jscs](http://jscs.info/) - Maintain JavaScript Code Style
* [jshint](http://jshint.com/) - Detect errors and potential problems in JavaScript code.

All IDE’s and Text Editors can handle these tools.

##### Documentation

To generates api documentation you can run gulp doc

#### Unit Testing

Unit testing can be launched manually with **karma start**.

~~It’s advisable to use~~ [~~expect.js~~](https://github.com/Automattic/expect.js) ~~instead of assert.~~

After investigate and testing the [expect.js](https://github.com/Automattic/expect.js) it don’t support some features for ES6, because this tool hasn’t activity at some time, that is why, it is recomended use the [chaijs](http://chaijs.com/) it is more versatile and have expect.js (but updated) and others tools that can be useful;

#### Karma

if you have some problems starting the karma tests, try running this commands for the following order:

1. npm uninstall karma karma-browserify karma-mocha karma-mocha-reporter karma-chrome-launcher -g
2. npm install karma-cli -g
3. npm install
4. jspm update

##### Note

This repository is ready to start working on development of runtime-core. The code will go to the **src** folder. The unit tests will be on **test** folder, following the name standard .spec.js

To run karma tests is mandatory to run **live-server** because of the mock-up’s dependencies: live-server --port=4000

#### Gulp Tasks

* [Documentation](#documentation)
* [Dist](#dist)
* [Build](#build)
* [Encode](#encode)

##### Documentation

Generate all documentation associated to runtime core;

* if you run **gulp doc** the documentation based on jsdoc3 will be generated on folder docs/jsdoc and you can interact;

gulp doc

* if you run **gulp api** the documentation is generate based on docs/api/ html files, and converted to markdown;

gulp api

* if you run **gulp docx** should be generated an .docx file, but **this process should be optimized**, is not working very well;

gulp docx

##### Dist

To distribute the runtime-core, you can make a distribution file.

Run the command:

// compact true | false;  
gulp dist --compact=false

##### Build

To distribute the runtime-core, but with the source code maps, and to detect where is some error.

Run the command: gulp build

##### Encode

In this repository, we have some tasks which can help you. If you need change some resource file, like an Hyperty or ProtoStub, and load it to the Hyperties.json or ProtoStubs.json, run the following command, and answer to the questions;

gulp compile --file=path/to/file;

### Example

This repository have a folder with an working example of Connector Hyperty (WebRTC call) and Group Chat Hyperty through Vertx Messaging Node;

### Notes

It was done an version of RuntimeCatalogue for local instances, based on the RuntimeCatalogue, and is activated by default;

## dev-runtime-browser

### Overview

This repository contains the code necessary to execute the reTHINK runtime core in a browser. reTHINK runtime core can also be executed in other Javascript runtimes such as Node.js.

The execution of the core runtime takes place in an iFrame which isolates it from the main application runtime (the window where the App javascript code is being executed). The only way to transmit messages between the main window and the iFrame is through the postMessage() method. This way, main application javascript code can not interact with the reTHINK runtime.

Addtionally to the iFrame, all the hyperties and protoStub will be executed as independient Web Workers (which will extend the sandBox class from the dev-core-runtime repository). This way we keep Hyperties and protoStub runtimes not directly accessible from the core runtime but using also the postMessage() mechanism.

### User view

#### Setup Environment

##### Configure jspm access to runtime-core repo

1. generate token with public\_repo permission enabled
2. Then execute the command below and you’ll be asked for the credentials:

* jspm registry config github

##### Configure dependencies

npm install -g jspm karma-cli gulp-cli  
 npm install  
 jspm install

or

npm run init-setup

#### Example of use

This repository have a folder with an use example of rethink.js. It initializes runtime and then you can use the console to invoke:

* rethink.requireHyperty(hypertyDescriptor);
* rethink.requireProtostub(domain);

To run the demo on example folder: - you need **live-server** running in the root folder. live-server --port=4000 - in your browser access to http://localhost:4000/example.

#### Distributable files

* rethink.js. It is the public interface. It exposes two methods:
  + requireHyperty(hypertyDescriptor)
  + requireProtoStub(domain)
* Moreover it create the ContextApp, where the hyperties are loaded and the iframe where the core is loaded.
* context-core.js. It includes the runtime core (RuntimeUA, MiniBus and SandBox) and defines browser implementations of SandboxFactory, SandboxWorker and SandboxApp. SandboxWorker and SandboxApp are responsibles for manage communication between boundaries.
* context-service.js It is responsible for load hyperties and protostub like ContextApp but in an isolated way.

### Developer view

#### How does it work?

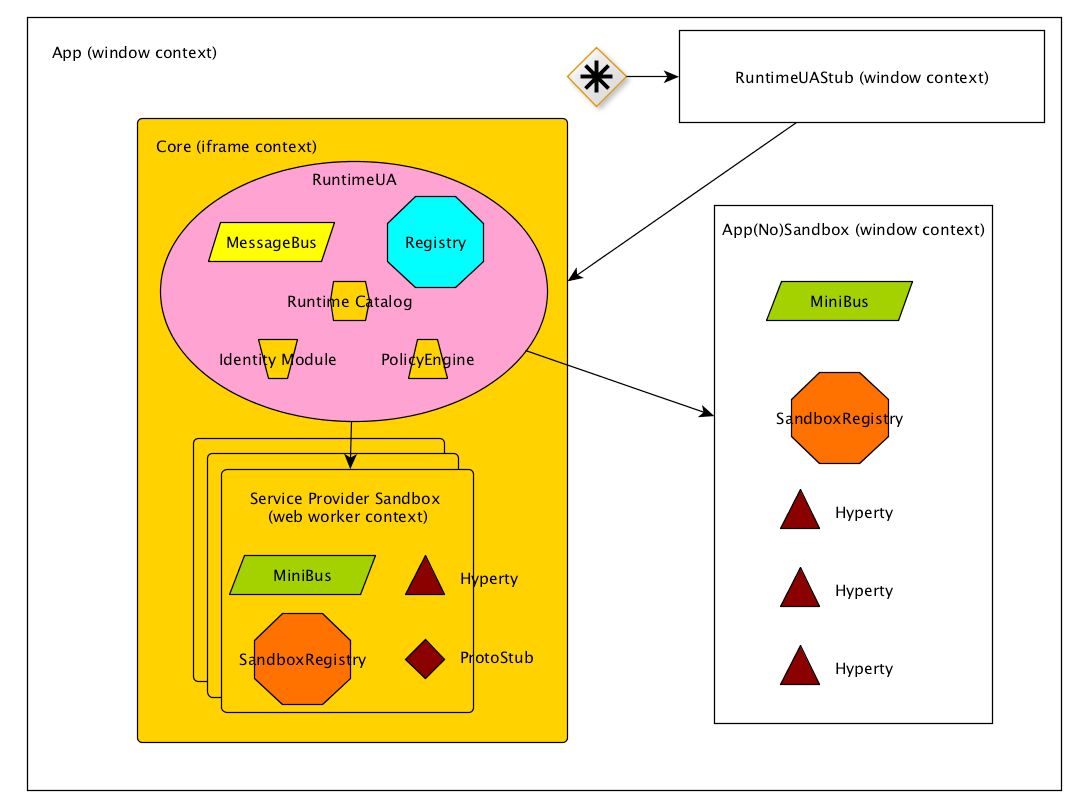


Figure - Runtime Browser

#### RuntimeUAStub responsibilities:

1. Expose loadHyperty and loadProtoStub to **client app**.
2. if Core Sandbox doesn’t exist it creates Core Sandbox.
3. Route messages from client app to core and vice versa.
4. Create **AppSandbox** when RuntimeUA set it. Virtually AppSandbox is created by RuntimeUA, but due to AppSandbox is running in the window context it should be created by RuntimeUAStub. RuntimeUA will send a message asking it to RuntimeUAStub.

#### Core/Service Provider Sandbox responsibilities:

1. Isolate RuntimeUA from client app.
2. Manage all the communication from and to internal components.

#### AppSandbox

1. Manage all the communication from and to internal components.

#### Unit Testing

Unit testing can be launched manually with **npm test**.

#### Javascript Environment

JavaScript code should be written in ES6. There are direct dependencies from nodejs and npm, these can be installed separately or in conjunction with [nvm](https://github.com/creationix/nvm)

#### Dependencies

* nodejs
* npm
* karma - Make the communication between unit test tool and jenkins. See more on [karma](http://karma-runner.github.io/0.13/index.html)
* mocha - Unit test tool. See more on [http://mochajs.org](http://mochajs.org/)
* jspm - Don’t need compile the code, it uses babel (or traucer or typescript) to run ES6 code on browser. Know more in [jspm.io](http://jspm.io/)
* gulp - Automate and enhance your workflow. See more about gulp on [gulp](http://gulpjs.com/)

#### Code Style and Hinting

On the root directory you will find **.jshintrc**, this file is a helper to maintain syntax consistency, it signals syntax mistakes and makes the code equal for all developers.

* [jshint](http://jshint.com/) - Detect errors and potential problems in JavaScript code.

All IDE’s and Text Editors can handle these tools.

## Standalone runtime application

### Introduction

The client side of the reTHINK architecture has been designed to be executed in a device which can execute a Javascript runtime, typically a web browser. This allows to be able to access to servcies provided through the reTHINK network from almost any device. Nowadays it is possible to run web browser in almost any personal gadget, however there may be devices where either is it not possible to run a browser or the available browsers does not suport the APIs required by the reTHINK browser runtime. For example, the browsers in iOS does not currently support the WebRTC API.

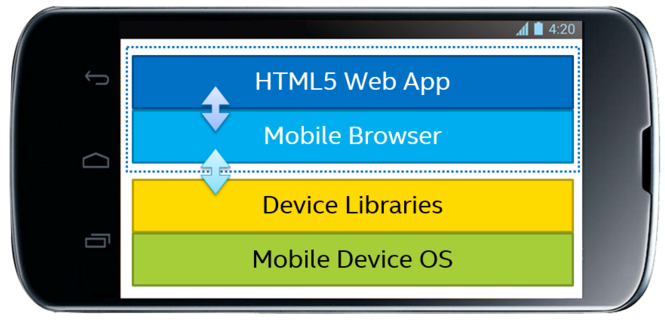


Figure - Web App executed in browser

That is the main reason why the creation of an application which can run the reTHINK client applications has been identified as a need. The use of web applications embedded in native application or even replacing them has become a common practice in the last years. This allows to re-use all the code developed for web applications therefore reducing the cost and time-to-market of new applications.

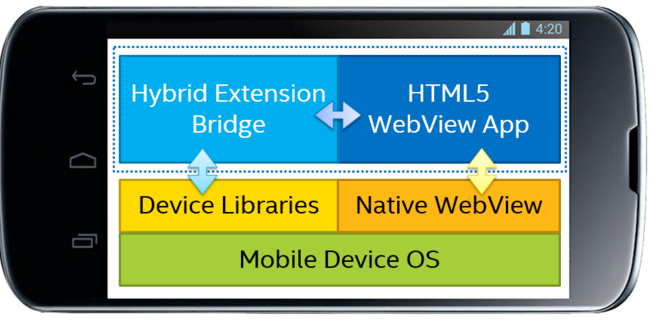


Figure - Hybrid App

There are several alternatives to execute web applications as native apps. In Android there webview elements directly provided by the OS and there are projects which allows to create native apps for both iOS and Android. For reTHINK the Crosswalk Project has been chosen to implement the native apps.

### Crosswalk Project

reTHINk standalone application allows to execute reTHINK runtime in Android and iOS devices without the need of having installed a browser will full support of the required APIs. The standalone application is based on the [Crosswalk Project](https://crosswalk-project.org/) from Intel. Crosswalk Project is an HTML application runtime, built on open source foundations, which extends the web platform with new capabilities. Crosswalk gives a web runtime for mobile and desktop applications. The immediate benefit of bundling an application with the Crosswalk webview is that everywhere the application runs, it uses the same, Chromium-based runtime. It is possible to create webviews for Android and iOS, but also for Windows and Linux Desktop applications so it makes any web application usable in almost any platform. In reTHINK only standalone runtime aplpication swiil be created for Android and iOS, as it always possible to install browsers which can execute reTHINK applications in Desktops.

WebRTC APIs are available in Crosswalk 5 or later on ARM; and Crosswalk 7.36.154.6 or later for x86. Web workers (also required for the browser runtime) is also supported by Crosswalk since previous versions.

### Android standalone application

The diagram below shows the architecture of the appplication. The hybrid application is created with Cordova which allows to access different sensors and services of the phone through a Javascript API. Cordova connects the App the Crosswalk Webview which is the part of the code which implements the WebRTC stack. Crosswalk will give a consistent Webview implementation across all the Android versions and it will guarantee that the reTHINK runtime will be executed correctly.

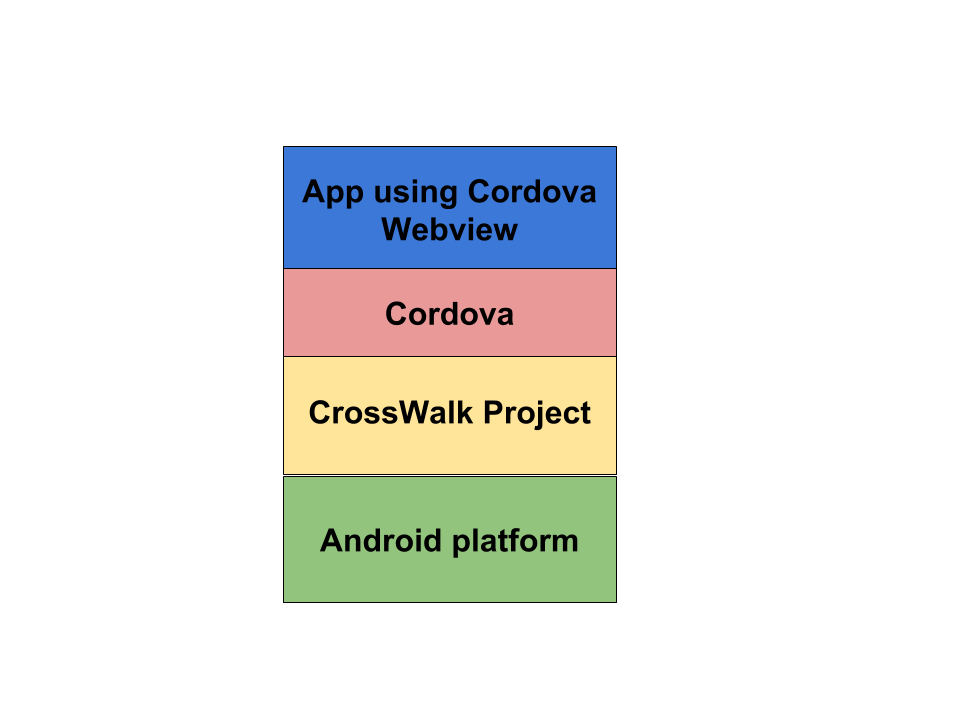


Figure - Standalone Android App

### Building the reTHINK Android application

#### Installing prerequisites

**openjdk-7:**

sudo apt-get install openjdk-7-jdk

**ant:**

sudo apt-get install ant

**android sdk:**

cd /opt/

curl -O http://dl.google.com/android/android-sdk\_r22.6.2-linux.tgz

./android-sdk-linux/tools./android

This last command will install the default list of packages. This process may take quite a while.

#### Building the application

It is necessary to indicate the public URL of the Web App which is going to be loaded in the hybrid App. In next releases it may be possible to load the HTML5/CSS/JS files in the own App.

make RETHINKWEBAPPURI=“http://…”

source build.env

make

#### Build the standalone application with Eclipse

It is also possible to build the standalone application using Eclipse. The general steps to build the application are included below:

1. Launch eclipse
2. Import xwalk-core-library project (3rdparty/xwalk\_core\_library/)
3. Import Cordova project (3rdparty/crosswalk-cordova-android/framework/)
4. Import standalone-ios project
5. Set the URL of the web application using reTHINK framework to be executed in the runtime application.
6. Build standalon-ios project as an Android application

### iOS standalone application

In iOS the architecture is slightly different from the Android architecture. Cordova is also use to build the application but the WebRTC stack will be provided by the eface2face plugin which includes a complete WebRTC library. In the Android App this is provided by Crosswalk. In iOS a complete simulation of the official WebRTC stack is provided by [!cordova-plugin-iosrtc](https://github.com/eface2face/cordova-plugin-iosrtc). It implementes the official [!WebRTC W3C API](https://www.w3.org/TR/webrtc/) and includes a compiled library with all the WebRTC code. The rest of the Javascript APIs that are required to execute the reTHINK runtime will be provided by Cordova.

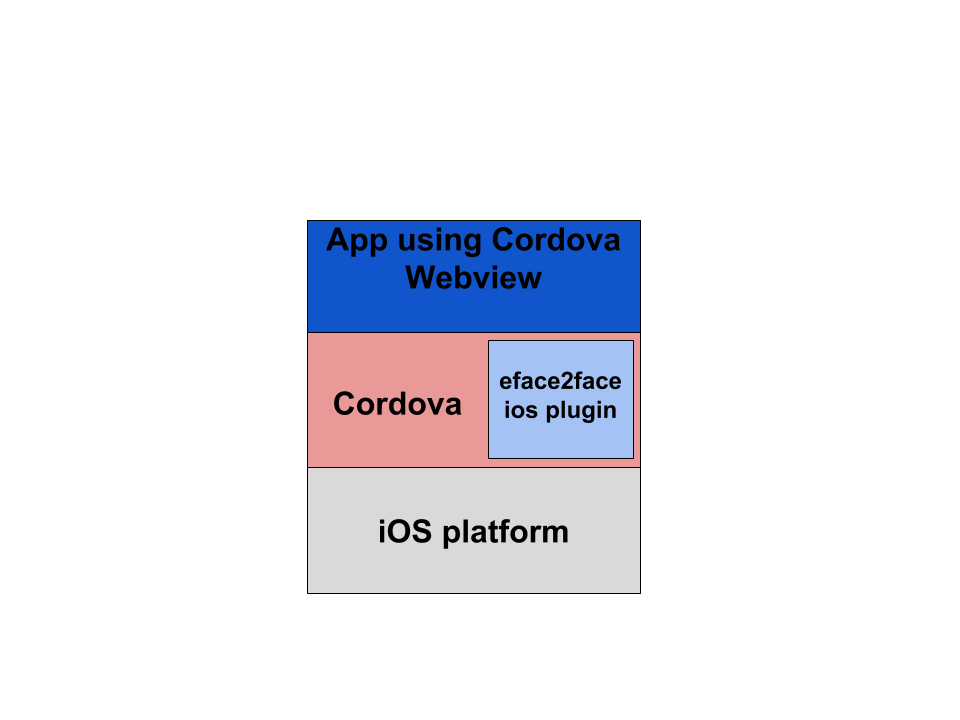


Figure - Standalone iOS App

#### Building reTHINK iOS standalone application

##### Requirements

1. OSX with XCode 5.
2. A valid Apple ID must be used (load associated certificates and profiles).

##### Build process

1. It is necessary to clone ios-rethink-standalone repository (this repository has not been yet created at the time of this writing) .
2. Open application project with XCode: sippo-ios/app/Sippo.xcodeproj.
3. Set the target location: change default values defined at Root.plist file (Settings.bundle->Root.plist in XCode project explorer).
4. Build application.

## vertx.io based Message Node (VertxMN)

### Overview

#### Functional location in the reTHINK Architecture

The vertx.io based Message Node is one of the reference implementations of the Message Node component in the reTHINK Architecture. The overall role of Message Nodes in the reTHINK Architecture is described in detail in [Hyperty Messaging Framework](https://github.com/reTHINK-project/dev-service-framework/blob/d3.2-working-docs/docs/manuals/hyperty-messaging-framework.md).

A general documentation and guideline for the development of Message nodes is given in [Message Nodes and Protostubs Development](https://github.com/reTHINK-project/dev-service-framework/blob/d3.2-working-docs/docs/manuals/development-of-protostubs-and-msg-nodes.md).

#### Dependencies

One of the responsibilities of Message Nodes in the reTHINK architecture is to perform the interactions with the Domain registry. Runtimes send special messages to the Message Nodes to register or query hyperties or data objects at the domain registry. The Message Nodes have to perform the interactions with the registry component and return the results back to the Runtime.

There is a need to have the domain registry running and configured. However, there is no specific initiation sequence. The procedures to achieve this are described in the following section.

### User View

This chapter provides instructions for the setup, configuration and operation of the Vertx Message Node as a docker container. The Docker file is already available in the repository.

#### 1. Installation of Git and Docker

You need to set up the following requirements. - [docker](https://docs.docker.com/) - [git client tool](https://git-scm.com/downloads) This is already a Docker dependency. If already installed you may skip it. - Test Docker setup. Fire the Docker shell and run docker run hello-world

#### 2. Cloning the repository and running Docker image

git clone https://github.com/reTHINK-project/dev-msg-node-vertx.git  
cd dev-msg-node-vertx

#### 3. Config VertxMN domain

The VertxMN is pointing at default domain ua.pt, but if other domain is needed it can be configured in node.config.json. Change an already entry, like dev, or create a new one. Config entry is selected with an environment variable MSG\_NODE\_CONFIG.

#### 4. Build and run Docker

docker build -t vertx-msg-node .  
docker run -it -e "MSG\_NODE\_CONFIG=dev" -p 9090:9090 vertx-msg-node

**Verify** if the VertxMN is running at docker host port map https://192.168.99.100:9090/, should return **Hello**. It’s ok at first to have an invalid certificate. The pre-configured self-signed certificate is pointing at host msg-node.ua.pt You can config your host OS file and add a new line for 192.168.99.100 msg-node.ua.pt.

#### 5. Testing

To fire the test suite you need **NPM** and all installed dependencies. - [NodeJS](https://nodejs.org/en/) will install NPM.npm install - Run 2 instances of the VertxMN to test cluster modes. You can run 2 dockers with diferent port maps -p 9090:9090 and -p 9091:9090. - For unit test it’s required to have the correct **host OS file** configured to msg-node.ua.pt and domain at **ua.pt** in the selected entry for node.config.json. - Run **karma start**

**create user** and **read user** will fail if there is no domain registry running.

### Developer view

Once the VertxMN is active, we are able to connect with the ProtoStub. The best example of how this is done is in the test/VertxProtoStub.js in “runtime connectivity” test. It’s important to send the “runtimeURL” in the config parameter, because it will be used to link the connection channel to the runtime.

With this it’s possible to send messages between runtimes, but Hyperty registration is something that should be done externally.

The connection is auto managed. It means, there is no need to call “connect()” explicitly, and it will always try to be in “connected” until “disconnect()” is called. Status messages are sent to “runtimeProtoStubURL/status”.

#### 1. Development dependencies

If the development is made without Docker, aditional dependencies are needed. - [Maven](https://maven.apache.org/install.html). - [Java 8 SDK](http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html). - JAVA\_HOME environment variable pointing to java path

#### 2. Structure of the project

The “dev-msg-node-vertx” GitHub repository is structured as follows:

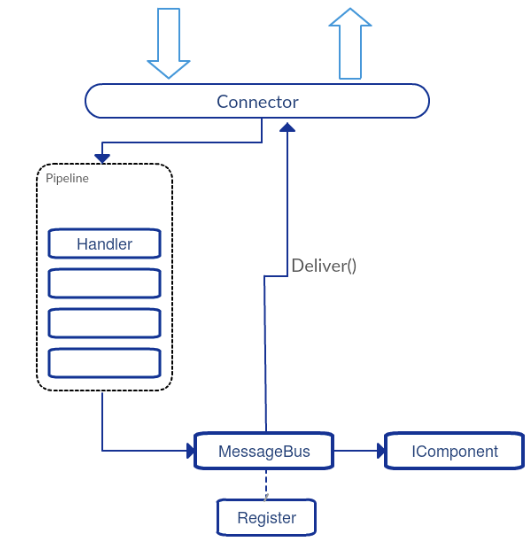
* ./Dockerfile … Docker file to build image.
* ./src/main/java/\*\* … The VertxMN source code
* ./src/js/client/\*\* … The sources for the Protocol stub
* ./test/\*\* … Test cases for the VertxMN
* ./target/ … Output for distribution files

#### 3. Build and run

Build a VertxMN distribution jar, executing mvn package. Build a VertxProtoStub distribution file, executing gulp build.

Run VertxMN with mvn exec:java -Dexec.args="9090". This will download all jar dependencies and run.

#### 4. Internal Architecture and components



Figure

There are 2 types of components that can integrate in the VertxMN implementation.\* Addressable, based in one destination address. Messages are deliver based on the “msg.to” field of the message.\* Interceptors that can intercept and verify every message that enters the Message Node, whatever the destination address.

##### Addressable Components

These are implementations of the interface IComponent extends Handler<PipeContext>, and are added to the MessageNode with the method PipeRegistry.installComponent(IComponent component). The only difference on the interface (between IComponent and Handler<PipeContext>) is an additional method to get the component address name, used for EventBus registration.

##### Interceptor Components

These are implementations of Handler<PipeContext>, and are added to the pipeline with Pipeline.addHandler(Handler<PipeContext> handler).

#### Use of PipeContext

Both types receive a PipeContext in the **handle** method when a message should be processed by the component. PipeContext gives access to the message with the getMessage() method, but also provides other useful methods like:\* next() method used in Interceptors that order the pipeline to execute the next interceptor. If no other interceptor exits, a delivery is proceeded.\* deliver() used internally by the pipeline, but can be also used to ignore all other pipeline handlers and deliver the message directly to the component that has the address of “msg.to”.\* fail(String from, String error) interrupts the pipeline flow and sends an error message back to the original “msg.from”. The “msg.from” of the reply is configured with the first parameter.\* reply(PipeMessage reply) does nothing to the pipeline flow and sends a reply back to original resource channel. Other similar and useful methods exists: replyOK(String from) and replyError(String from, String error)\* disconnect() order the underlying resource channel to disconnect.

## Matrix.org based Message Node (MatrixMN)

### Overview

#### Functional location in the reTHINK Architecture

The Matrix.org based Message Node is one of the reference implementations of the Message Node component in the reTHINK Architecture. The overall role of Message Nodes in the reTHINK Architecture is described in detail in [Hyperty Messaging Framework](https://github.com/reTHINK-project/dev-service-framework/blob/d3.2-working-docs/docs/manuals/hyperty-messaging-framework.md).

A general documentation and guideline for the development of Message nodes is given in [Message Nodes and Protostubs Development](https://github.com/reTHINK-project/dev-service-framework/blob/d3.2-working-docs/docs/manuals/development-of-protostubs-and-msg-nodes.md).

#### Dependencies

One of the responsibilities of Message Nodes in the reTHINK architecture is to perform the interactions with the Domain registry. Runtimes send special messages to the Message Nodes to register or query hyperties or data objects at the domain registry. The Message Nodes have to perform the interactions with the registry component and return the results back to the Runtime.

For that reason the Matrix Message Node has a dependency from the domain registry component. That means, before a MatrixMN can be used successfully, an instance of the domain registry must be running. The procedures to achieve this are described in the following section.

### User View

This chapter provides instructions for the setup, configuration and operation of the Matrix Message Node as a docker container. Assuming you are running a standard Debian 8 Jessie the following steps can be used to setup the environment. Other distributions my need a different setup.

#### 1. Installation of NodeJS and Docker

You need to set up the following requirements. - [nodejs 5.x](https://nodejs.org/en/download/package-manager/#debian-and-ubuntu-based-linux-distributions) - nodejs-legacy for Debian (Ubuntu might not need this) - [docker](https://docs.docker.com/engine/installation/debian/) - If the docker daemon cannot be reached you need to run sudo usermod -aG docker $USER. After that logout and back in or use this command su - $USER. - If the test sudo docker run hello-world fails you may need a different kernel. Some kernels like those provided by OVH are not working with docker.

#### 2. Installation of repository-tools and cloning the repository

Execute these commands to install the needed tools and dependencies.

sudo npm install -g gulp  
git clone https://github.com/reTHINK-project/dev-msg-node-matrix.git  
cd dev-msg-node-matrix  
npm install

#### 3. Building the MatrixMN

Afterwards you can build the MatrixMN distribution. Please make sure you are located in the dev-msg-node-matrix directory. Simply type pwd to check that. Then run the following commands.

rm -rf dist && gulp build && gulp dist  
cd dist/docker  
./build-docker-image.sh matrix1.rethink

The last parameter matrix1.rethink specifies the domain name for MatrixMN.

When errors occur while building MatrixMN which relate to 404 errors you might want to check your Docker DNS settings. Try editing /etc/default/docker and uncomment the line #DOCKER\_OPTS="--dns 8.8.8.8 --dns 8.8.4.4". You can also add the DNS servers of your company. The resulting line may look like this one DOCKER\_OPTS="--dns 8.8.8.8 --dns 8.8.4.4 --dns 10.1.100.252 --dns 10.1.100.246".

Build the image again and if the errors continue to show up you can check the /etc/resolv.conf file. It should have a line or lines containing something similar to search company.tld lan lan. or nameserver 10.1.100.252.

#### 4. Building the Registry

As described in the Overview section, the MatrixMN has a dependency to a domain registry, because it needs to interact with this Registry to register and read hyperties and data objects for user-ids.

Therefore the domain registry must be built and started before the MatrixMN can be used. Please change to the dev-registry-domain/server directory after cloning it from https://github.com/reTHINK-project/dev-registry-domain.git in a place of your choice. Then run:

docker build -t dev-registry-domain .  
docker images

Now you should see the 2 docker images which were built.

#### 5. Starting the Registry

The first image to be started is the registry.

#cd to dev-msg-node-matrix/dist/docker  
./startregistry.sh

#### 6. Starting the MatrixMN

Open another terminal and execute the following.

#cd to dev-msg-node-matrix/dist/docker  
./start.sh

The MatrixMN will now start which might take a while. You can check whether it is finished by executing > docker logs dev-msg-node-matrix

and looking for the last line being similar to: > synapse.storage.TIME - 212 - INFO - - Total database time: 0.000% {get\_all\_pushers(0): 0.000%,

##### 7. Testing

Finally you can test the correctness of the setup.

#cd to dev-msg-node-matrix or a subdirectory  
gulp test

The test will attempt to open the google chrome browser. If none of the test are executed you might need to install it with sudo apt-get install chromium-browser.

### Developer view

The MatrixMN code does not modify any Matrix.org specific code. It only implements additional components that can be attached to an untouched Matrix Homeserver (HS). This additional code is written in JavaScript, which is executed in a nodejs runtime.

#### Suggested documentation

Detailed information about the main concepts of the Matrix.org infrastructure can be found in this high-level [**Matrix-Overview**](./Matrix-Overview.md).

In order to understand the internal architecture of the MatrixMN the documentation at [**MatrixMN-internal-architecture**](./MatrixMN-internal-architecture.md) is suggested.

#### Structure of the GitHub repository

The “dev-msg-node-matrix” GitHub repository is structured as follows:

* **./src/mn** … The node.js sources for the MatrixMN
* **./src/stub** … The sources for the Protocol stub
* **./src/docker** … Scripts and additional files required for the setup of a dockerized version of the Matrix Message Node
* **./test** … Test cases for the Matrix Message Node

#### Development setup

To setup the repository for developments on the MatrixMN, follow first the steps described in the “User View” chapter before. With the resulting setup the MatrixMN code will be executed inside of the docker container. That means that for each change on the MN code the docker container must be restarted. Otherwise the changes take no effect. This is of course not very comfortable for coding and testing in short cycles.

To improve this situation, the MatrixMN can also be operated as a stand-alone NodeJS process outside of the docker container. However, this requires some manipulations on the setup of the container.

Since the MatrixMN operates as an Application Service (AS) for the Matrix HomeServer, the HomeServer must be able to address the AS for sending requests. If the AS is in the same container this address is always *localhost*. If we run it outside this does not work anymore.

Following steps must be performed to make it work:

1. Identify the hosts address on the docker bridge. Execute:

* ip a
* and search for the ip address corresponding to the “docker0” bridge interface (may be named similar)

1. Edit ./src/mn/rethink-mn-registration.yaml. Replace localhost with this ip-address.
2. Execute step 3 (Building the MN) of the installation instructions
3. Start the Matrix docker container without the MatrixMN code

* #cd to dist/docker  
  ./startdevelopment.sh

1. execute the MatrixMN stand-alone

* gulp startmn

Now you can perform changes and extension on the Matrix MN implementation (below directory ./src/mn) and do a

gulp build && gulp dist && gulp startmn

whenever you want to test your changes without the need to restart the full docker container.

If your development is done, change the configuration in rethink-mn-registration.yaml back to “localhost”, stop the docker container, perform step 3 again and start the container with the built-in MatrixMN code.

### Matrix.org - Overview and core concepts

The Matrix mission statement (from [matrix.org spec](https://matrix.org/speculator/spec/head/intro.html)): > *The end goal of Matrix is to be a ubiquitous messaging layer for synchronising arbitrary data between sets of people, devices and services - be that for instant messages, VoIP call setups, or any other objects that need to be reliably and persistently pushed from A to B in an interoperable and federated manner.*

##### Homeservers

The core components of the Matrix architecture are the Home Servers (HS). Each Homeserver is responsible for one domain. Each client connects to one HS, wich is responsible for the own domain. Communication between different domains is supported by built-in federation mechanisms that sync and maintain the history of shared communication sessions among the domains. Home Servers use normal DNS to find, resolve and contact each other. The Federation API between Homeservers is based on HTTPs and therefore encrypted and secured by default.

The reference implementation of a Matrix HS, called *Synapse*, is written in Python and available on GitHub [Synapse](https://github.com/matrix-org/synapse).

##### Clients

Matrix clients connect to a HomeServer by using a REST-based [Client-Server-API](http://matrix.org/docs/spec/r0.0.1/client_server.html). Clients can either implement the corresponding REST calls directly or choose to use one of the SDK’s, which are available for a lot of different systems and programming languages, including Android, IOS, Python, NodeJS etc. These SDK’s abstract the REST API and provide a lot of high-level convenience methods.

Following picture shows the main data flow in a federated matrix architecture.



Figure - Main data flow in a matrix architecture

As this Figure shows, clients just connect to their own HS, but due to the built-in federation between the Homeservers they can communicate with Matrix clients from other domains just out-of-the-box. This allows to form a Matrix Eco-System, just by adding Homeservers to the public Internet and make their domains resolvable via DNS.

##### The Matrix Room concept

The Matrix.org project was influenced by concepts from traditional Instant Messaging systems. This can be seen especially in the concept of “communication rooms”, which behave like a chat room. This concept implies that *every* communication requires a room. Even for a single message from one client to a dedicated receiver a room must be created first and the receiver must have joined this room in order to receive this message. Rooms are persistent. They can be re-entered after successive login sessions.

##### Application services

An Application Service is an implementation of a special service function that can be attached to a Homeserver. Based on certain patterns, messages are filtered and forwarded to the Application Service that performs application specific tasks. This concept is quite comparable to Application Servers in the IP Multimedia Subsystem (IMS) framework. It can, for example, be used for aggregation and accounting purposes, but also for the implementation of “breakout” communication to other types of messaging infrastructures it fits well.

Like the corresponding concept in IMS, also Matrix Application Services operate in a special trust-relationship with the HS. This trusted state allows them to listen to messages that match special user-/or room-name patterns as well as to create users on-the-fly and to operate on behalf of them.  
It must be noted that Application services do (until now) only play a passive role. They can listen to messages, but they can (by-design) not block or modify them.

#### MatrixMN internal architecture

##### General considerations, requirements and decisions

Matrix.org is a very vital and active project with frequent releases and new surrounding developments and projects. However, the provided API’s are rather stable and backward compatible as seen so far. For that reason the first and most important decision for the design of the Matrix based messaging node was made: - Don’t touch the core of the Homeserver implementation! Just implement components that use standard API’s to add the required functionalities!

If the reTHINK concepts of Protocol Stubs and Messaging Nodes are translated to the Matrix.org concepts, it seems like the Stubs map well to Matrix clients and the Messaging Nodes to Matrix Homeservers.

As described before, a Matrix client communicates to a Homeserver via a REST protocol. Several available SDKs encapsulate this protocol, so that - at the end - it is not obvious for the implementor of a client, when and how much traffic is generated between client and HS. Furthermore the SDKs come with a set of dependencies that potentially blow up the size of a Stub and make its deployment more complicated.

For these reasons it was decided that: - The stub should be kept as small and simple as possible to ensure easy deployment. - The real Matrix REST communication should be limited to the Messaging Node internally while the communication protocol between Stub and Messaging Node can be implemented differently.

Another challenge is that Matrix.org requires provisioned users and established room relationships between them to perform a communication. The establishment of a room relationship between two users is a process that requires several round-trip message exchanges between their corresponding clients and rather complex state monitoring. This process can take potentially too long and might lead to unacceptable delays for an ad-hoc message exchange between two Runtimes. Furthermore this would produce a lot of persistent room relationships in the Matrix Homeservers which might never be re-used again.

Therefore following requirements for the design of the Matrix based Messaging Node were identified:

* The Messaging node must support an automatic provisioning of Matrix users on-the-fly.
* The message routing must not depend on fully established room relationships between the matrix users that correspond with the “from” and “to” addresses of a retHINK message.

These 5 decisions guided the architecture approach that is described in the following section.

##### Matrix Messaging Node architecture

The figure below illustrates the high-level architecture of the Matrix Messaging Node. The Matrix Homeserver itself is left unchanged. It was decided to wrap all additional functionalities into an Application Service that communicates via standardized APIs with the HS.

The reTHINK AS includes a WebSocket Server component that is the endpoint for connection requests from Protocol Stubs, which are deployed and running in reTHINK Runtimes. The Matrix Client - Manager is then responsible to identify the connected Stubs/Runtimes and to maintain the life-cycle of a dedicated Matrix Client instance for this particular Stub-/Runtime-connection. This includes the instantiation and also the re-assignment of Matrix Clients to stubs, in case of re-connections, for instance after a network interruption.

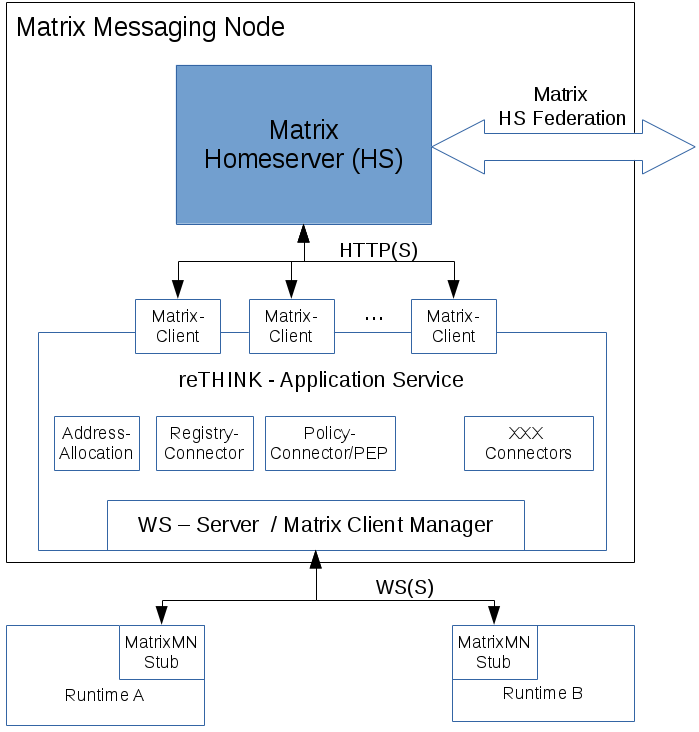
-

Figure - matrix-address-allocation

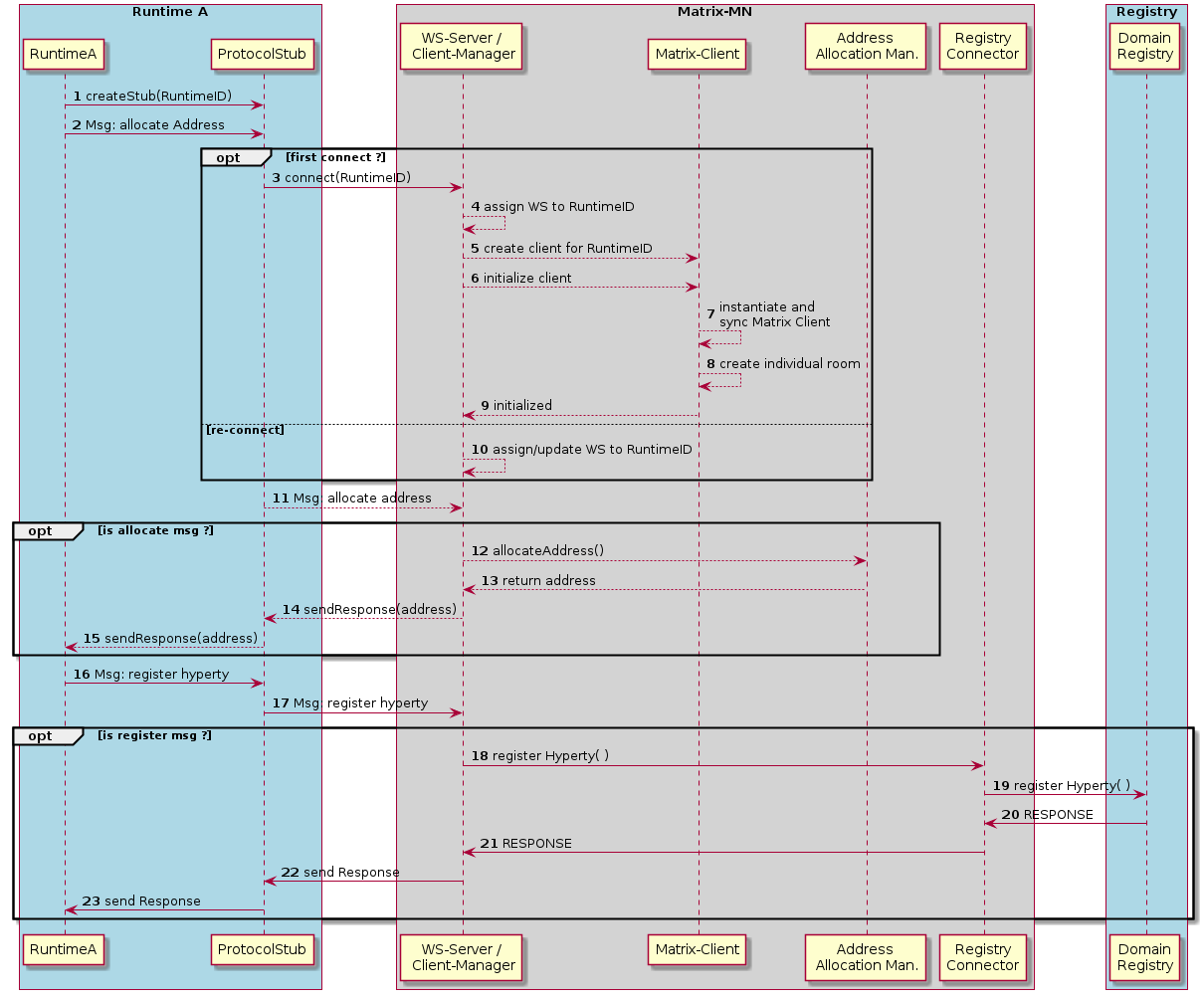
Due to the trust-relationship of the Application Service, the Matrix Client Manager is allowed to create and auto-provision Matrix Clients on-the-fly. These Matrix Clients then communicate with the HS core via standard client API’s.

The Address-Allocation Manager handles CREATE requests to the address-allocation part of the Messaging Node. It creates addresses that can be used to address Hyperties in the connected Runtimes and keeps their relation to the connected Stub.

The Registry connector handles requests to fetch user data from the domain registry or to register Hyperties with the allocated addresses. Furthermore the Policy Enforcement Point component allows to block message flows according to policies which can be managed from remote via the Policy Management Connector. Additional connectors can be implemented and used to manage or control certain aspect of the message routing.

##### Dynamic Views

The following sequence chart shows the processes for the connection of a Protocol Stub at the Messaging Node, the allocation of a hyperty address and the registratio of this address at the domain registry.



Figure

- 1-2: The Stub is initialized and connected by the Runtime (simplified) and the Runtime sends a CREATE message to the address-allocation module of the Messaging Node.

* 3: The Stub connects with the WS-Server for the first time and provides the RuntimeID as identifier.
* 4: The WS-Server assigns the given RuntimeID to the Websocket.
* 5-6: The Client Manager creates and inititializes a Matrix Client for the given RuntimeID.
* 7-8: The Matrix Client inititializes (connects to the HS, syncs etc.) and creates its individual room.
* 10: In case of a re-connection, the Client Manager only updates the relation between the existing client and the new Websocket
* 11: The WS-Server receives the address allocation request from the Stub
* 12 -15: The Address Allocation Manager is invoked to create a new address, which is returned to the Runtime via the Protocol Stub
* 16-21: These steps show the flow for a message to create a hyperty registration. It invokes the Registry Connector that interacts with the domain registry to perform the registration. The response will be returned via the corresponding stub to the Runtime

In order to avoid expensive creation of bi-lateral room relationships and to allow to block messages depending on policies, it was decided that the AS acts as a “man-in-the-middle” between sender and receiver. To achieve this, each Matrix Client creates a private individual room with no other invited or joined members during its Initialization.

The alias name of these rooms starts with a defined prefix “#\_rethink\_”. The AS is configured to monitor such rooms. Every message that arrives via the Stub will then only be sent to the individual room of the sender. The AS receives the message and can perform the required policy decisions. If everything is OK, then the message is forwarded to the room that corresponds to the user in the “to”-field of the message. The Matrix client, that is the owner of the receivers individual room, receives the message and will send it to the receivers Runtime via the connected Stub.

The following sequence chart illustrates this routing principle.

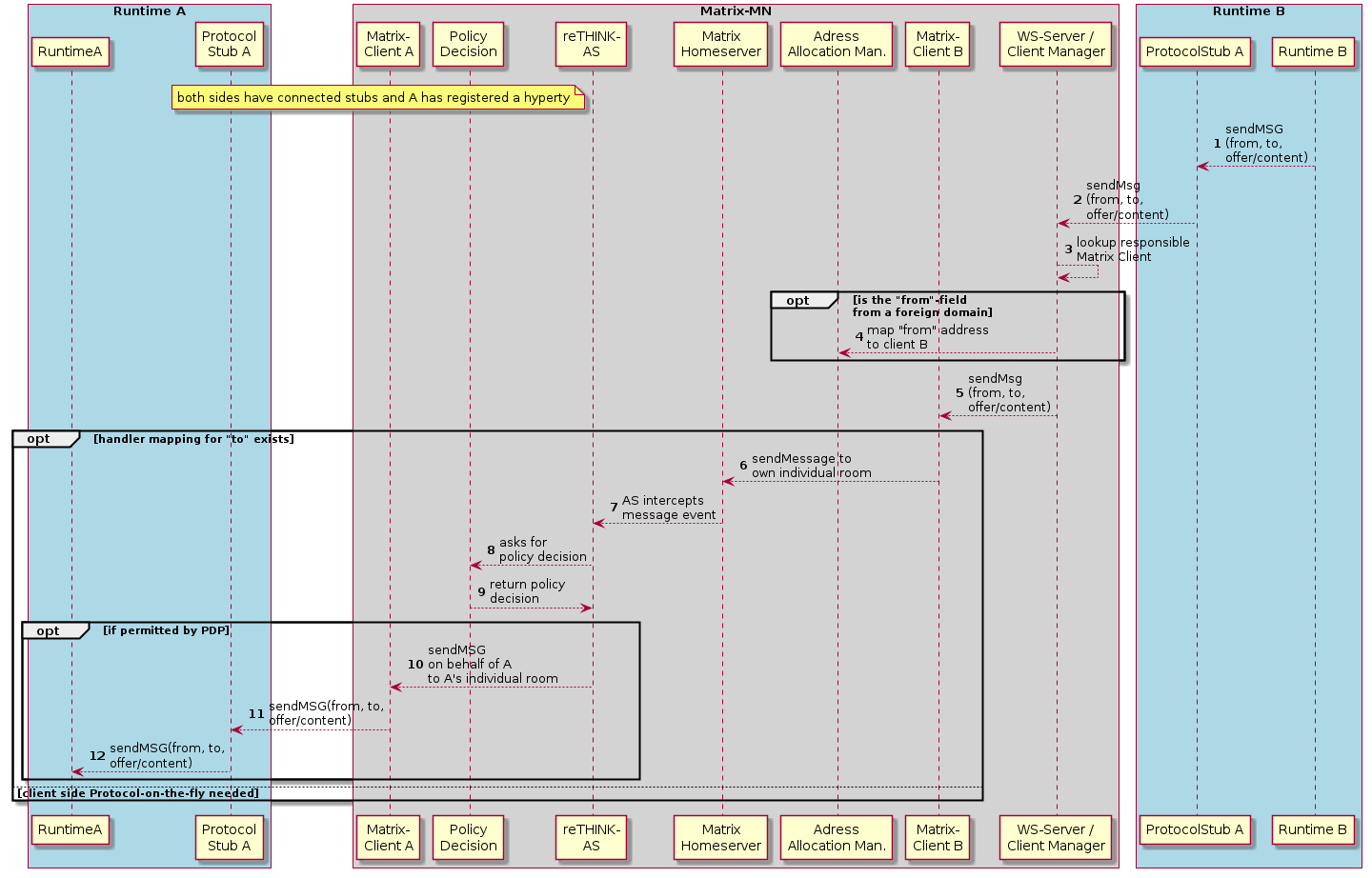


Figure - matrix-message-routing

This sequence starts with the precondition that both sides (Runtimes A and B) have connected a stub with the Matrix Messaging Node, that Runtime A has registered a Hyperty and Runtime B knows its address.

* 1-2: Runtime B sends a message (e.g. an call invitation) via the Stub to the Websocket Server
* 3: The Client Manager looks up the Matrix-Client that is responsible for this Runtime
* 4: If the “from”-field is not for the own domain, a mapping of this “from” address to the Matrix Client is created. This is required to investigate the return route in requests that have been initiated from a foreign domain.
* 6: The client Manager forwards the message to the client that is responsible for B’s connected Stub and this one sends it to its own individual room.
* 7-9: The reTHINK Application Server receives the message, because it monitors the individual rooms and invokes the policy component to get a decision about the further handling of the message.
* 10: In case of a positive decision, the AS sends the message to client A’s individual room.
* 11-12: The message is forwarded via A’s Protocol stub to Runtime A

#### Matrix Messaging Stub

Due to the described architecture where the real Matrix REST communication is kept on the server side, the protocol stub can be kept very small and simple. It implements a Websocket client that automatically connects to the WS-Server as soon as a message is going to be sent. During the connection establishment it forwards the RuntimeID, that it was created for, to the Server, so that the Messaging Node can identify and correctly assign this stub also after potentially interruptions and re-connections.

The Messaging Stub is integrated with the Runtimes Messaging Bus. Each message that is received via the Websocket is forwarded to the bus and will be routed there to the correct receiver.

Furthermore the Stub uses the Bus to publish events about its internal status, especially on changes of its connection state.

## NodeJS based Message Node

### Overview

The NodeJS based Message Node is one of the reference implementations of the Message Node component in the reTHINK Architecture.

Like other Message Nodes, it has responsibilities to perform messages delivering between different hyperties. And by design, it interact with other Rethink components like the domain registry or runtime.

So, that implies to have an running instance of the domain registry to get the nodejs message node running correctly.

You will find a general documentation and guideline Message nodes Development in [Message Nodes and Protostubs Development](https://github.com/reTHINK-project/dev-service-framework/blob/d3.2-working-docs/docs/manuals/development-of-protostubs-and-msg-nodes.md).

### User View

#### Setup Environment

This documentation does not provide a OS dependant instructions : NodeJS message node can be used on any OS compatible with redis & nodejs tools. A dockerfile is provided, so it can be integrated in a docker instance as well.

##### Javascript Environment

JavaScript code should be written in ES6.

Please follow instructions on [official nodejs installation documentation](https://nodejs.org/en/download/package-manager/) to setup the NodeJS environnement.  
This include the npm manager for node modules.

##### dependencies:

* nodejs
* npm
* karma - A simple tool that allows you to execute JavaScript code in multiple real browsers. See more on [karma](http://karma-runner.github.io/0.13/index.html)
* mocha - Unit test tool. See more on [http://mochajs.org](http://mochajs.org/)
* gulp - Automate and enhance your workflow. See more about gulp on [gulp](http://gulpjs.com/)

##### Quick start

On the first time you are cloning this repository, you need to run the command  
$ **npm run init-setup**;

After running successfully this command you will have 2 folders (node\_modules and vendor), these folders are excluded from the commit process, and are only for development.

Check the server configuration file for custom setting (url, port, …) :

Now start server with command :  
$ **node src/main/server.js**;

You should see a notice like that :  
[Date] [INFO] server - [S] HTTP & WS server listening on 9090

if you already have the project configured on your machine, you only need run the command npm install to update package & new dependencies.

#### Service architecture

The figure below illustrates the service architecture of the NodeJS Messaging Node.

Combine with node redis sentinel client, each node share session datas with each others through redis storage.  
Redis-Sentinel monitor & notify redis cluster of data change between nodejs instance.



Figure - NodeJS & Redis clustering using Redis-Sentinel

For security consideration, it’s advized to use a proxy (as describe in the following scheme) in front of node instance to not give direct access to nodejs instance. It’s recommanded to use NGinx server for that ([from NGiNX](https://www.nginx.com/blog/nginx-nodejs-websockets-socketio/), [from Socket.io](http://socket.io/docs/using-multiple-nodes/)).  
By the way it also provide a good load balancer solution (HAProxy is another good one).



Figure - Web proxy in front of node instances

#### Hyperty development

To use the message nodes in client side, please refer to [Hyperty development tutorial](https://github.com/reTHINK-project/dev-service-framework/blob/d3.2-working-docs/docs/manuals/development-of-hyperties.md)

### Developer View

#### Repository structure

This repository is ready to start working on development.  
The code will go to the **src** folder, it contains also the main server script in src/main/ folder.

The unit tests will be on **test** folder, following the name standard .spec.js

Server (config.js) & tools (gulp, karma, etc…) configuration is located in root folder.

#### Code Style and Hinting

On the root directory you will find **.jshintrc** and **.jscsrc**, these files are helpers to maintain syntax consistency, it signals syntax mistakes and makes the code equal for all developers.

* [jscs](http://jscs.info/) - Maintain JavaScript Code Style
* [jshint](http://jshint.com/) - Detect errors and potential problems in JavaScript code.

Most IDEs and Text Editors can handle these tools.

#### Documentation

To generates api documentation you can run :

$ **gulp doc**  
This will generate HTML documentation in docs/ folder.

#### Unit Testing

We use Karma test runner to execute mocha test.

To run unit test, you need first to lauch a server node with command :

$ **node src/main/server.js**  
… then start karma test runner (from main directory) :  
$ **karma start**

Karma will launch the browser (chromium in this case) to execute all tests in test/ folder and show result in console. Tests are automatically redone when code is modified.

#### Server components

##### NodeJS

###### Socket.io

Socket.io is a well-known library that provide real-time bidirectionnal event-based communication.  
It able to handle the connection transparently for developpers :

* the protocol negociation (long-polling, websocket,etc…) with client depending of network capabilities
* connection always on with heartbeat packets
* message broadcasting
* session datas
* clustering consideration with multiple data storage drivers

###### ExpressJS

Express.js is a minimalist web framework commonly used in front of socket.io server.  
It provide a robust set of features for web and mobile applications, like request routing, and a solid stack for third-party middleware.

##### Redis

Redis is an in-memory data structure store, used as database, cache and message broker. It supports various type of data structures such as string, hashes, lists… It have a persistent mode, but it’s mainly used to store temporary datas like session or connection information.

Redis has built-in replication, and provides high availability via Redis Sentinel and automatic partitioning with Redis Cluster.

#### Core components

This section describe the functional blocks of the Messaging Node architecture.

The graphic below describe message event processing with components. 

Figure

##### Entry point

Msg node start with server.js script that read configuration from config.js and instanciate <> class.

This unique class initialize main components and start listening for incoming websocket client. On each new protostub connection, socket.io events are bind to <> instance associated with socket ressource.

##### Registry

A global Registry class is used by MsgNode to manage internal components and configuration. It allow internal component to share reference to configuration and others components.

##### SessionManager

The SessionManager class handle client connection state change. ###### Note Link with identity service ?

##### Message bus

MessageBus class provide a message system that publish information to all components.

###### Note

/! Redis bus manager is not implemented yet, so message cannot be broadcast in a msg node cluster : code in place allow only to publish message through current node instance.

##### Address allocation management

The class AddressAllocationManager handle hyperty URLs allocation once client ask for registration.

###### Note

/! For the moment, foreign hyperty instance pool are not managed. Link with global domain registry ?

##### Message

On each message received from protostub, MsgNode built a ClientMessage instance containing Message, and dispatch to Client instance. It’s also used as container to built reply on client.

1. APIs Documentation

# AddressAllocation

Class will ask to the message node for addresses

## Constructor Summary

Public Constructor public

[constructor](../../../class/src/registry/AddressAllocation.js~AddressAllocation.html#instance-constructor-constructor)(url: URL.URL, bus: [MiniBus](../../../class/src/bus/MiniBus.js~MiniBus.html))

Create an Address Allocation

## Member Summary

Public Members public get

[url](../../../class/src/registry/AddressAllocation.js~AddressAllocation.html#instance-get-url): [string](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String): \*

get the URL value

## Method Summary

Public Methods public

[create](../../../class/src/registry/AddressAllocation.js~AddressAllocation.html#instance-method-create)(domain: Domain, number: [number](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number)): [Promise](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise)<HypertyURL>

Ask for creation of a number of Hyperty addresses, to the domain message node.

## Public Constructors

### public constructor(url: URL.URL, bus: [MiniBus](../../../class/src/bus/MiniBus.js~MiniBus.html)) [source](../../../file/src/registry/AddressAllocation.js.html#lineNumber17)

Create an Address Allocation

#### Params:

Name Type Attribute Description url URL.URL url from who is sending the message

bus [MiniBus](../../../class/src/bus/MiniBus.js~MiniBus.html) MiniBus used for address allocation

## Public Members

### public get url: [string](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String): \* [source](../../../file/src/registry/AddressAllocation.js.html#lineNumber31)

get the URL value

#### Return:

|  |  |
| --- | --- |
| string</spa n> | The url value; |

## Public Methods

### public create(domain: Domain, number: [number](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number)): [Promise](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise)<HypertyURL> [source](../../../file/src/registry/AddressAllocation.js.html#lineNumber39)

Ask for creation of a number of Hyperty addresses, to the domain message node.

#### Params:

Name Type Attribute Description domain Domain Domain of the message node.

number [number](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number) Number of addresses to request

#### Return:

|  |  |
| --- | --- |
| Promise</s pan><HypertyURL&g t; | A list of HypertyURL’s |

Generated by [ESDoc(0.4.5)](https://esdoc.org) [source](../../../file/src/graphconnector/BloomFilter.js.html#lineNumber38)

# Bus

#### Direct Subclass:

[MessageBus](../../../class/src/bus/MessageBus.js~MessageBus.html), [MiniBus](../../../class/src/bus/MiniBus.js~MiniBus.html)

#### Indirect Subclass:

[Sandbox](../../../class/src/sandbox/Sandbox.js~Sandbox.html)

## Constructor Summary

Public Constructor public

[constructor](../../../class/src/bus/Bus.js~Bus.html#instance-constructor-constructor)

## Method Summary

Public Methods public

[addListener](../../../class/src/bus/Bus.js~Bus.html#instance-method-addListener)(url: URL, listener: Listener): MsgListener

Register listener to receive message when “msg.to === url”.

public

[addResponseListener](../../../class/src/bus/Bus.js~Bus.html#instance-method-addResponseListener)(url: URL, msgId: [number](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number), responseListener: [Function](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function))

Manually add a response listener.

public

[bind](../../../class/src/bus/Bus.js~Bus.html#instance-method-bind)(outUrl: URL, inUrl: URL, target: [MiniBus](../../../class/src/bus/MiniBus.js~MiniBus.html)): Bound

Helper method to bind listeners (in both directions) into other MiniBus target.

public

[postMessage](../../../class/src/bus/Bus.js~Bus.html#instance-method-postMessage)(msg: Message, responseCallback: [Function](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function)): [number](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number)

Send messages to local listeners, or if not exists to external listeners.

public

[removeAllListenersOf](../../../class/src/bus/Bus.js~Bus.html#instance-method-removeAllListenersOf)(url: URL)

Remove all existent listeners for the URL

public

[removeResponseListener](../../../class/src/bus/Bus.js~Bus.html#instance-method-removeResponseListener)(url: URL, msgId: [number](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number))

Remove the response listener.

## Public Constructors

### public constructor [source](../../../file/src/bus/Bus.js.html#lineNumber17)

## Public Methods

### public addListener(url: URL, listener: Listener): MsgListener [source](../../../file/src/bus/Bus.js.html#lineNumber35)

Register listener to receive message when “msg.to === url”. Special url “\*” for default listener is accepted to intercept all messages.

#### Params:

Name Type Attribute Description url URL Address to intercept, tha is in the message “to”

listener Listener listener

#### Return:

|  |  |
| --- | --- |
| MsgListener | instance of MsgListener |

### public addResponseListener(url: URL, msgId: [number](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number), responseListener: [Function](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function)) [source](../../../file/src/bus/Bus.js.html#lineNumber57)

Manually add a response listener. Only one listener per message ID should exist. ATENTION, there is no timeout for this listener. The listener should be removed with a removeResponseListener, failing to do this will result in a unreleased memory problem.

#### Params:

Name Type Attribute Description url URL Origin address of the message sent, “msg.from”.

msgId [number](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number) Message ID that is returned from the postMessage.

responseListener [Function](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function) Callback function for the response

### public bind(outUrl: URL, inUrl: URL, target: [MiniBus](../../../class/src/bus/MiniBus.js~MiniBus.html)): Bound [source](../../../file/src/bus/Bus.js.html#lineNumber85)

Helper method to bind listeners (in both directions) into other MiniBus target.

#### Params:

Name Type Attribute Description outUrl URL Outbound URL, register listener for url in direction “this -> target”

inUrl URL Inbound URL, register listener for url in direction “target -> this”

target [MiniBus](../../../class/src/bus/MiniBus.js~MiniBus.html) The other target MiniBus

#### Return:

|  |  |
| --- | --- |
| Bound | an object that contains the properties [thisListener, targetListener] and the unbind method. |

### public postMessage(msg: Message, responseCallback: [Function](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function)): [number](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number) [source](../../../file/src/bus/Bus.js.html#lineNumber200)

Send messages to local listeners, or if not exists to external listeners. It’s has an optional mechanism for automatic management of response handlers. The response handler will be unregistered after receiving the response, or after response timeout (default to 3s).

#### Params:

Name Type Attribute Description msg Message Message to send. Message ID is automatically added to the message.

responseCallback [Function](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function) Optional parameter, if the developer what’s automatic response management.

#### Return:

|  |  |
| --- | --- |
| number</spa n> | Returns the message ID, in case it should be needed for manual management of the response handler. |

### public removeAllListenersOf(url: URL) [source](../../../file/src/bus/Bus.js.html#lineNumber74)

Remove all existent listeners for the URL

#### Params:

Name Type Attribute Description url URL Address registered

### public removeResponseListener(url: URL, msgId: [number](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number)) [source](../../../file/src/bus/Bus.js.html#lineNumber66)

Remove the response listener.

#### Params:

Name Type Attribute Description url URL Origin address of the message sent, “msg.from”.

msgId [number](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number) Message ID that is returned from the postMessage

Generated by [ESDoc(0.4.5)](https://esdoc.org) [source](../../../file/src/utils/EventEmitter.js.html#lineNumber5)

# GlobalRegistryRecord

Represents the user’s information for the global registry.

## Constructor Summary

Public Constructor public

[constructor](../../../class/src/graphconnector/GlobalRegistryRecord.js~GlobalRegistryRecord.html#instance-constructor-constructor)

Constructs a new empty object.

## Member Summary

Public Members public

[userIDs](../../../class/src/graphconnector/GlobalRegistryRecord.js~GlobalRegistryRecord.html#instance-member-userIDs): \*

## Method Summary

Public Methods public

[getRecord](../../../class/src/graphconnector/GlobalRegistryRecord.js~GlobalRegistryRecord.html#instance-method-getRecord)(): [Object](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object)

Constructs a new object representing information about one contact.

## Public Constructors

### public constructor [source](../../../file/src/graphconnector/GlobalRegistryRecord.js.html#lineNumber10)

Constructs a new empty object.

## Public Members

### public userIDs: \* [source](../../../file/src/graphconnector/GlobalRegistryRecord.js.html#lineNumber13)

## Public Methods

### public getRecord(): [Object](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object) [source](../../../file/src/graphconnector/GlobalRegistryRecord.js.html#lineNumber25)

Constructs a new object representing information about one contact.

#### Return:

|  |  |
| --- | --- |
| Object</spa n> | object A JavaScript Object with all fields for the Global Registry Record. |

Generated by [ESDoc(0.4.5)](https://esdoc.org) [source](../../../file/src/graphconnector/GraphConnector.js.html#lineNumber14)

# GraphConnector

The Graph Connector contains the contact list/address book.

## Constructor Summary

Public Constructor public

[constructor](../../../class/src/graphconnector/GraphConnector.js~GraphConnector.html#instance-constructor-constructor)(HypertyRuntimeURL: [string](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String), MessageBus: [messageBus](../../../class/src/graphconnector/GraphConnector.js~GraphConnector.html#instance-get-messageBus))

Constructs a new and empty Graph Connector.

## Member Summary

Public Members public

[contacts](../../../class/src/graphconnector/GraphConnector.js~GraphConnector.html#instance-member-contacts): \*

public

[contactsBloomFilter1Hop](../../../class/src/graphconnector/GraphConnector.js~GraphConnector.html#instance-member-contactsBloomFilter1Hop): \*

public

[globalRegistryRecord](../../../class/src/graphconnector/GraphConnector.js~GraphConnector.html#instance-member-globalRegistryRecord): \*

public

[groups](../../../class/src/graphconnector/GraphConnector.js~GraphConnector.html#instance-member-groups): \*

public

[lastCalculationBloomFilter1Hop](../../../class/src/graphconnector/GraphConnector.js~GraphConnector.html#instance-member-lastCalculationBloomFilter1Hop): \*

public get

[messageBus](../../../class/src/graphconnector/GraphConnector.js~GraphConnector.html#instance-get-messageBus)(messageBus: [MessageBus](../../../class/src/bus/MessageBus.js~MessageBus.html)): \*

Returns the MessageBus.

public set

[messageBus](../../../class/src/graphconnector/GraphConnector.js~GraphConnector.html#instance-set-messageBus)(messageBus: [MessageBus](../../../class/src/bus/MessageBus.js~MessageBus.html)): \*

Sets the MessageBus.

## Method Summary

Public Methods public

[addContact](../../../class/src/graphconnector/GraphConnector.js~GraphConnector.html#instance-method-addContact)(guid: [string](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String), firstName: [string](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String), lastname: [string](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String))

Add a contact to the Graph Connector.

public

[addUserID](../../../class/src/graphconnector/GraphConnector.js~GraphConnector.html#instance-method-addUserID)(userID: [string](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String))

Adds a UserID for the user.

public

[calculateBloomFilter1Hop](../../../class/src/graphconnector/GraphConnector.js~GraphConnector.html#instance-method-calculateBloomFilter1Hop)()

Calculates the Bloom filter containing all non-private contacts.

public

[checkGUID](../../../class/src/graphconnector/GraphConnector.js~GraphConnector.html#instance-method-checkGUID)(guid: [string](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String)): array

Checks, if the given GUID is known and returns a list of contacs that are direct connections as well as a list of contacts that (most likely) know the given contact.

public

[generateGUID](../../../class/src/graphconnector/GraphConnector.js~GraphConnector.html#instance-method-generateGUID)(): [string](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String)

Generates a GUID and returns a mnemonic from which the GUID can be re-created later.

public

[getContact](../../../class/src/graphconnector/GraphConnector.js~GraphConnector.html#instance-method-getContact)(name: [string](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String)): array

Gets contacts by name.

public

[queryGlobalRegistry](../../../class/src/graphconnector/GraphConnector.js~GraphConnector.html#instance-method-queryGlobalRegistry)(guid: [string](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String)): [Promise](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise)

Queries the Global Registry for a given GUID.

public

[removeContact](../../../class/src/graphconnector/GraphConnector.js~GraphConnector.html#instance-method-removeContact)(guid: [string](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String))

Remove a contact from the Graph Connector.

public

[removeUserID](../../../class/src/graphconnector/GraphConnector.js~GraphConnector.html#instance-method-removeUserID)(userID: [string](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String))

Removes a UserID for the user.

public

[sendGlobalRegistryRecord](../../../class/src/graphconnector/GraphConnector.js~GraphConnector.html#instance-method-sendGlobalRegistryRecord)(jwt: [string](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String)): Propmise

Takes the Global Registry Record as a signed JWT and sends it to the Global Registry via the MessageBus.

public

[signGlobalRegistryRecord](../../../class/src/graphconnector/GraphConnector.js~GraphConnector.html#instance-method-signGlobalRegistryRecord)(): [string](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String)

SignGenerates a public/private key pair from a given mnemonic.

public

[useGUID](../../../class/src/graphconnector/GraphConnector.js~GraphConnector.html#instance-method-useGUID)(mnemonicAndSalt: [string](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String)): [Promise](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise)

Generates a public/private key pair from a given mnemonic (16 words).

## Public Constructors

### public constructor(HypertyRuntimeURL: [string](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String), MessageBus: [messageBus](../../../class/src/graphconnector/GraphConnector.js~GraphConnector.html#instance-get-messageBus)) [source](../../../file/src/graphconnector/GraphConnector.js.html#lineNumber23)

Constructs a new and empty Graph Connector.

#### Params:

Name Type Attribute Description HypertyRuntimeURL [string](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String) The Hyperty Runtime URL.

MessageBus [messageBus](../../../class/src/graphconnector/GraphConnector.js~GraphConnector.html#instance-get-messageBus) The Message Bus.

## Public Members

### public contacts: \* [source](../../../file/src/graphconnector/GraphConnector.js.html#lineNumber24)

### public contactsBloomFilter1Hop: \* [source](../../../file/src/graphconnector/GraphConnector.js.html#lineNumber368)

### public globalRegistryRecord: \* [source](../../../file/src/graphconnector/GraphConnector.js.html#lineNumber27)

### public groups: \* [source](../../../file/src/graphconnector/GraphConnector.js.html#lineNumber30)

### public lastCalculationBloomFilter1Hop: \* [source](../../../file/src/graphconnector/GraphConnector.js.html#lineNumber25)

### public get messageBus(messageBus: [MessageBus](../../../class/src/bus/MessageBus.js~MessageBus.html)): \* [source](../../../file/src/graphconnector/GraphConnector.js.html#lineNumber43)

Returns the MessageBus.

### public set messageBus(messageBus: [MessageBus](../../../class/src/bus/MessageBus.js~MessageBus.html)): \* [source](../../../file/src/graphconnector/GraphConnector.js.html#lineNumber51)

Sets the MessageBus.

## Public Methods

### public addContact(guid: [string](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String), firstName: [string](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String), lastname: [string](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String)) [source](../../../file/src/graphconnector/GraphConnector.js.html#lineNumber331)

Add a contact to the Graph Connector.

#### Params:

Name Type Attribute Description guid [string](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String) GUID of the new contact.

firstName [string](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String) First name of the new contact.

lastname [string](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String) Last name of the new contact.

### public addUserID(userID: [string](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String)) [source](../../../file/src/graphconnector/GraphConnector.js.html#lineNumber300)

Adds a UserID for the user.

#### Params:

Name Type Attribute Description userID [string](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String) The UserID for a Domain Registry to add for the user.

### public calculateBloomFilter1Hop() [source](../../../file/src/graphconnector/GraphConnector.js.html#lineNumber358)

Calculates the Bloom filter containing all non-private contacts.

### public checkGUID(guid: [string](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String)): array [source](../../../file/src/graphconnector/GraphConnector.js.html#lineNumber393)

Checks, if the given GUID is known and returns a list of contacs that are direct connections as well as a list of contacts that (most likely) know the given contact.

#### Params:

Name Type Attribute Description guid [string](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String) GUID of the contact to look for.

#### Return:

|  |  |
| --- | --- |
| array | relatedContacts List of related direct contacts and of related friends-of-friends contacts.The format is: RelatedContacts<Direct<GraphCo nnectorContactData>,FoF<GraphC onnectorContactData>>. |

### public generateGUID(): [string](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String) [source](../../../file/src/graphconnector/GraphConnector.js.html#lineNumber59)

Generates a GUID and returns a mnemonic from which the GUID can be re-created later.

#### Return:

|  |  |
| --- | --- |
| string</spa n> | mnemonic A string with 16 words. |

### public getContact(name: [string](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String)): array [source](../../../file/src/graphconnector/GraphConnector.js.html#lineNumber377)

Gets contacts by name.

#### Params:

Name Type Attribute Description name [string](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String) First or last name to look for in the contact list.

#### Return:

|  |  |
| --- | --- |
| array | matchingContacts Contacts matching the given name. The format is: Contacts<GraphConnectorContactDat a>. |

### public queryGlobalRegistry(guid: [string](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String)): [Promise](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise) [source](../../../file/src/graphconnector/GraphConnector.js.html#lineNumber254)

Queries the Global Registry for a given GUID. Returns a Graph Connector Contact Data as a Promise.

#### Params:

Name Type Attribute Description guid [string](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String) The GUID to query the Global Registry for

#### Return:

|  |  |
| --- | --- |
| Promise</s pan> | Promise Graph Connector Contact Data containing UserIDs. |

### public removeContact(guid: [string](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String)) [source](../../../file/src/graphconnector/GraphConnector.js.html#lineNumber343)

Remove a contact from the Graph Connector.

#### Params:

Name Type Attribute Description guid [string](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String) GUID of the user to be removed.

### public removeUserID(userID: [string](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String)) [source](../../../file/src/graphconnector/GraphConnector.js.html#lineNumber317)

Removes a UserID for the user.

#### Params:

Name Type Attribute Description userID [string](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String) The UserID to remove.

### public sendGlobalRegistryRecord(jwt: [string](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String)): Propmise [source](../../../file/src/graphconnector/GraphConnector.js.html#lineNumber214)

Takes the Global Registry Record as a signed JWT and sends it to the Global Registry via the MessageBus. Returns the response code of the REST-interface of the Global Registry as a Promise.

#### Params:

Name Type Attribute Description jwt [string](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String) The Global Registry Record as a signed JWT.

#### Return:

|  |  |
| --- | --- |
| Propmise | Promise Response Code from Global Registry. |

### public signGlobalRegistryRecord(): [string](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String) [source](../../../file/src/graphconnector/GraphConnector.js.html#lineNumber198)

SignGenerates a public/private key pair from a given mnemonic.

#### Return:

|  |  |
| --- | --- |
| string</spa n> | JWT JSON Web Token ready to commit to Global Registry. |

### public useGUID(mnemonicAndSalt: [string](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String)): [Promise](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise) [source](../../../file/src/graphconnector/GraphConnector.js.html#lineNumber92)

Generates a public/private key pair from a given mnemonic (16 words). Expects a string containing 16 words seperated by single spaces. Retrieves data from the Global Registry.

#### Params:

Name Type Attribute Description mnemonicAndSalt [string](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String) A string of 16 words.

#### Return:

|  |  |
| --- | --- |
| Promise</s pan> | Promise Global Registry Record. |

Generated by [ESDoc(0.4.5)](https://esdoc.org) [source](../../../file/src/graphconnector/GraphConnectorContactData.js.html#lineNumber5)

# GraphConnectorContactData

Represents information about a contact.

## Constructor Summary

Public Constructor public

[constructor](../../../class/src/graphconnector/GraphConnectorContactData.js~GraphConnectorContactData.html#instance-constructor-constructor)(guid: [String](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String), firstName: [String](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String), lastName: [String](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String))

Constructs a new object representing information about one contact.

## Member Summary

Public Members public set

[contactsBloomFilter1Hop](../../../class/src/graphconnector/GraphConnectorContactData.js~GraphConnectorContactData.html#instance-set-contactsBloomFilter1Hop)(bf: [BloomFilter](../../../class/src/graphconnector/BloomFilter.js~BloomFilter.html)): \*

Sets the friends-of-friends Bloom filter containing the hashed GUIDs of the contacts for the contact.

public get

[contactsBloomFilter1Hop](../../../class/src/graphconnector/GraphConnectorContactData.js~GraphConnectorContactData.html#instance-get-contactsBloomFilter1Hop): [BloomFilter](../../../class/src/graphconnector/BloomFilter.js~BloomFilter.html): \*

Returns the Bloom filter containing the hashed GUIDs of the contacts for the contact.

public get

[firstName](../../../class/src/graphconnector/GraphConnectorContactData.js~GraphConnectorContactData.html#instance-get-firstName): [String](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String): \*

Returns the first name.

public set

[firstName](../../../class/src/graphconnector/GraphConnectorContactData.js~GraphConnectorContactData.html#instance-set-firstName)(firstName: [String](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String)): \*

Sets the first name.

public set

[groups](../../../class/src/graphconnector/GraphConnectorContactData.js~GraphConnectorContactData.html#instance-set-groups)(groups: List<[String](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String)>): \*

Sets the groups.

public get

[groups](../../../class/src/graphconnector/GraphConnectorContactData.js~GraphConnectorContactData.html#instance-get-groups): List<[String](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String)>: \*

Returns the groups.

public get

[guid](../../../class/src/graphconnector/GraphConnectorContactData.js~GraphConnectorContactData.html#instance-get-guid): [String](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String): \*

Returns the GUID.

public set

[guid](../../../class/src/graphconnector/GraphConnectorContactData.js~GraphConnectorContactData.html#instance-set-guid)(GUID: [String](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String)): \*

Sets the GUID.

public set

[lastName](../../../class/src/graphconnector/GraphConnectorContactData.js~GraphConnectorContactData.html#instance-set-lastName)(lastName: [String](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String)): \*

Sets the last name.

public get

[lastName](../../../class/src/graphconnector/GraphConnectorContactData.js~GraphConnectorContactData.html#instance-get-lastName): [String](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String): \*

Returns the last name.

public set

[privateContact](../../../class/src/graphconnector/GraphConnectorContactData.js~GraphConnectorContactData.html#instance-set-privateContact)(boolPrivate: [Boolean](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Boolean)): \*

Sets the privacy status of the contact according to the given Boolean value.

public get

[privateContact](../../../class/src/graphconnector/GraphConnectorContactData.js~GraphConnectorContactData.html#instance-get-privateContact): [Boolean](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Boolean): \*

Returns the privacy status of the contact.

public get

[residenceLocation](../../../class/src/graphconnector/GraphConnectorContactData.js~GraphConnectorContactData.html#instance-get-residenceLocation): [String](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String): \*

Returns the geohash of the residence location.

public set

[residenceLocation](../../../class/src/graphconnector/GraphConnectorContactData.js~GraphConnectorContactData.html#instance-set-residenceLocation)(geohash: [String](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String)): \*

Sets the geohash of the residence location.

public set

[userIDs](../../../class/src/graphconnector/GraphConnectorContactData.js~GraphConnectorContactData.html#instance-set-userIDs)(userIDs: List<[String](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String)>): \*

Sets the userIDs.

public get

[userIDs](../../../class/src/graphconnector/GraphConnectorContactData.js~GraphConnectorContactData.html#instance-get-userIDs): List<[String](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String)>: \*

Returns the user IDs.

## Public Constructors

### public constructor(guid: [String](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String), firstName: [String](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String), lastName: [String](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String)) [source](../../../file/src/graphconnector/GraphConnectorContactData.js.html#lineNumber13)

Constructs a new object representing information about one contact.

#### Params:

Name Type Attribute Description guid [String](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String) The GUID of the new contact.

firstName [String](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String) The first name of the new contact.

lastName [String](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String) The last name of the new contact.

## Public Members

### public set contactsBloomFilter1Hop(bf: [BloomFilter](../../../class/src/graphconnector/BloomFilter.js~BloomFilter.html)): \* [source](../../../file/src/graphconnector/GraphConnectorContactData.js.html#lineNumber118)

Sets the friends-of-friends Bloom filter containing the hashed GUIDs of the contacts for the contact.

### public get contactsBloomFilter1Hop: [BloomFilter](../../../class/src/graphconnector/BloomFilter.js~BloomFilter.html): \* [source](../../../file/src/graphconnector/GraphConnectorContactData.js.html#lineNumber110)

Returns the Bloom filter containing the hashed GUIDs of the contacts for the contact.

#### Return:

|  |  |
| --- | --- |
| [BloomFilter](../../../class/s%20%20rc/graphconnector/BloomFilter.js~%20Blo%20omFilter.html) | bf Bloom filter for the contact. |

### public get firstName: [String](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String): \* [source](../../../file/src/graphconnector/GraphConnectorContactData.js.html#lineNumber62)

Returns the first name.

#### Return:

|  |  |
| --- | --- |
| String</spa n> | firstName First name of the contact. |

### public set firstName(firstName: [String](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String)): \* [source](../../../file/src/graphconnector/GraphConnectorContactData.js.html#lineNumber70)

Sets the first name.

### public set groups(groups: List<[String](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String)>): \* [source](../../../file/src/graphconnector/GraphConnectorContactData.js.html#lineNumber151)

Sets the groups.

### public get groups: List<[String](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String)>: \* [source](../../../file/src/graphconnector/GraphConnectorContactData.js.html#lineNumber143)

Returns the groups.

#### Return:

|  |  |
| --- | --- |
| List<[String](https://developer.mozilla.org/en-US/%20%20docs/Web/JavaScript/Reference/Glo%20bal%20_Objects/String)> | groups Groups of the contact. |

### public get guid: [String](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String): \* [source](../../../file/src/graphconnector/GraphConnectorContactData.js.html#lineNumber29)

Returns the GUID.

#### Return:

|  |  |
| --- | --- |
| String</spa n> | GUID GUID of the contact. |

### public set guid(GUID: [String](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String)): \* [source](../../../file/src/graphconnector/GraphConnectorContactData.js.html#lineNumber37)

Sets the GUID.

### public set lastName(lastName: [String](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String)): \* [source](../../../file/src/graphconnector/GraphConnectorContactData.js.html#lineNumber86)

Sets the last name.

### public get lastName: [String](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String): \* [source](../../../file/src/graphconnector/GraphConnectorContactData.js.html#lineNumber78)

Returns the last name.

#### Return:

|  |  |
| --- | --- |
| String</spa n> | lastName Last name of the contact. |

### public set privateContact(boolPrivate: [Boolean](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Boolean)): \* [source](../../../file/src/graphconnector/GraphConnectorContactData.js.html#lineNumber102)

Sets the privacy status of the contact according to the given Boolean value.

### public get privateContact: [Boolean](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Boolean): \* [source](../../../file/src/graphconnector/GraphConnectorContactData.js.html#lineNumber94)

Returns the privacy status of the contact.

#### Return:

|  |  |
| --- | --- |
| Boolean</s pan> | privateContact True/false value indicating the privacy status of the contact. |

### public get residenceLocation: [String](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String): \* [source](../../../file/src/graphconnector/GraphConnectorContactData.js.html#lineNumber127)

Returns the geohash of the residence location.

#### Return:

|  |  |
| --- | --- |
| String</spa n> | geohash Geohash of the residence location. |

### public set residenceLocation(geohash: [String](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String)): \* [source](../../../file/src/graphconnector/GraphConnectorContactData.js.html#lineNumber135)

Sets the geohash of the residence location.

### public set userIDs(userIDs: List<[String](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String)>): \* [source](../../../file/src/graphconnector/GraphConnectorContactData.js.html#lineNumber53)

Sets the userIDs.

### public get userIDs: List<[String](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String)>: \* [source](../../../file/src/graphconnector/GraphConnectorContactData.js.html#lineNumber45)

Returns the user IDs.

#### Return:

|  |  |
| --- | --- |
| List<[String](https://developer.mozilla.org/en-US/%20%20docs/Web/JavaScript/Reference/Glo%20bal%20_Objects/String)> | userIDs UserIDs of the contact. |

Generated by [ESDoc(0.4.5)](https://esdoc.org) [source](../../../file/src/registry/HypertyDiscovery.js.html#lineNumber7)

# HypertyDiscovery

Core HypertyDiscovery interface Class to allow applications to search for hyperties using the message bus

## Constructor Summary

Public Constructor public

[constructor](../../../class/src/registry/HypertyDiscovery.js~HypertyDiscovery.html#instance-constructor-constructor)(msgbus: [MessageBus](../../../class/src/bus/MessageBus.js~MessageBus.html), runtimeURL: RuntimeURL)

To initialise the HypertyDiscover, which will provide the support for hyperties to query users registered in outside the internal core.

## Method Summary

Public Methods public

[discoverHypertyPerUser](../../../class/src/registry/HypertyDiscovery.js~HypertyDiscovery.html#instance-method-discoverHypertyPerUser)(email: email): [Promise](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise)

function to request about users registered in domain registry, and return the hyperty instance if found.

## Public Constructors

### public constructor(msgbus: [MessageBus](../../../class/src/bus/MessageBus.js~MessageBus.html), runtimeURL: RuntimeURL) [source](../../../file/src/registry/HypertyDiscovery.js.html#lineNumber15)

To initialise the HypertyDiscover, which will provide the support for hyperties to query users registered in outside the internal core.

#### Params:

Name Type Attribute Description msgbus [MessageBus](../../../class/src/bus/MessageBus.js~MessageBus.html) msgbus

runtimeURL RuntimeURL runtimeURL

## Public Methods

### public discoverHypertyPerUser(email: email): [Promise](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise) [source](../../../file/src/registry/HypertyDiscovery.js.html#lineNumber29)

function to request about users registered in domain registry, and return the hyperty instance if found.

#### Params:

Name Type Attribute Description email email

#### Return:

|  |  |
| --- | --- |
| Promise</s pan> | Promise |

Generated by [ESDoc(0.4.5)](https://esdoc.org) [source](../../../file/src/registry/HypertyInstance.js.html#lineNumber7)

# IdentityModule

The Identity Module (Id Module) is the component responsible for handling the user identity and the association of this identity with the Hyperty instances, in order to make Hyperty instances identifiable. The identity in the reTHINK project is not fixed to a unique Identity Service Provider, but obtained through several different Identity sources. With this approach, the Id Module provides to the user the option to choose the preferred method for authentication. This module will thus able to support multiple Identity acquisition methods, such as OpenID connect 1.0, Kerberos System, or authentication through smart cards. For example, a user with a Google account can use the Google as an Identity Provider to provide Identity Tokens, which can be used by the Identity Module to associate it with a Hyperty instance.

The Identity Module uses a node package, the HelloJS, which is a client-side JavaScript API for authentication that facilitates the requests for the OpenID connect protocol. This method allows for some abstraction when making requests for different Identity Providers, such as OpenID connect used by Google, Facebook, Microsoft, for example.

When a request for a user identity is made using the method loginWithRP(identifier, scope), this method will analyse the Identity Provider chosen to obtain an identity and will use the HelloJS node package with the selected Identity Provider and identity scope. After the HelloJS request for an Access Token to the Identity Providers, the user will be prompted to authenticate towards the Identity Provider. Upon receiving the Access Token, this token is validated with a RESTful web service request to an endpoint on the Identity Provider Authorization Server, and after the validation is done, an ID token is obtained with the information according to the scope required. This ID token is then preserved in this module that can obtained through the getIdentities() and is passed as return value of the loginWithRP function. The methods generateAssertion and validateAssertion have not yet been developed.

## Constructor Summary

Public Constructor public

[constructor](../../../class/src/identity/IdentityModule.js~IdentityModule.html#instance-constructor-constructor)

This is the constructor to initialise the Identity Module it does not require any input.

## Method Summary

Public Methods public

[generateAssertion](../../../class/src/identity/IdentityModule.js~IdentityModule.html#instance-method-generateAssertion)(contents: DOMString, origin: DOMString, usernameHint: DOMString): IdAssertion

Generates an Identity Assertion

public

[getAssertionTrustLevel](../../../class/src/identity/IdentityModule.js~IdentityModule.html#instance-method-getAssertionTrustLevel)(assertion: DOMString)

Trust level evaluation of a received IdAssertion

public

[getIdentities](../../../class/src/identity/IdentityModule.js~IdentityModule.html#instance-method-getIdentities)(): [Array](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array)<Identities>

Function to return all the identities registered within a session by a user.

public

[loginWithRP](../../../class/src/identity/IdentityModule.js~IdentityModule.html#instance-method-loginWithRP)(identifier: Identifier, scope: Scope): [Promise](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise)

Function to request an ID Token from a user.

public

[registerIdentity](../../../class/src/identity/IdentityModule.js~IdentityModule.html#instance-method-registerIdentity)()

Register a new Identity with an Identity Provider

public

[registerWithRP](../../../class/src/identity/IdentityModule.js~IdentityModule.html#instance-method-registerWithRP)()

In relation with a classical Relying Party: Registration

public

[setHypertyIdentity](../../../class/src/identity/IdentityModule.js~IdentityModule.html#instance-method-setHypertyIdentity)()

In relation with a Hyperty Instance: Associate identity

public

[validateAssertion](../../../class/src/identity/IdentityModule.js~IdentityModule.html#instance-method-validateAssertion)(assertion: DOMString): [Promise](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise)

Function to validate an identity assertion generated previously.

## Public Constructors

### public constructor [source](../../../file/src/identity/IdentityModule.js.html#lineNumber36)

This is the constructor to initialise the Identity Module it does not require any input.

## Public Methods

### public generateAssertion(contents: DOMString, origin: DOMString, usernameHint: DOMString): IdAssertion [source](../../../file/src/identity/IdentityModule.js.html#lineNumber219)

Generates an Identity Assertion

#### Params:

Name Type Attribute Description contents DOMString contents

origin DOMString origin

usernameHint DOMString usernameHint

#### Return:

|  |  |
| --- | --- |
| IdAssertion | IdAssertion |

### public getAssertionTrustLevel(assertion: DOMString) [source](../../../file/src/identity/IdentityModule.js.html#lineNumber241)

Trust level evaluation of a received IdAssertion

#### Params:

Name Type Attribute Description assertion DOMString assertion

### public getIdentities(): [Array](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array)<Identities> [source](../../../file/src/identity/IdentityModule.js.html#lineNumber61)

Function to return all the identities registered within a session by a user. These identities are returned in an array containing a JSON package for each user identity.

#### Return:

|  |  |
| --- | --- |
| Array< /span> <Identities> | Array Identities |

### public loginWithRP(identifier: Identifier, scope: Scope): [Promise](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise) [source](../../../file/src/identity/IdentityModule.js.html#lineNumber76)

Function to request an ID Token from a user. If no token exists, the Identity Module will try to obtain one from an Identity Provider, and the user will be asked to authenticate towards the Identity Provider. The function returns a promise with a token containing the user information.

#### Params:

Name Type Attribute Description identifier Identifier identifier

scope Scope scope

#### Return:

|  |  |
| --- | --- |
| Promise</s pan> | Promise IDToken containing the user information |

### public registerIdentity() [source](../../../file/src/identity/IdentityModule.js.html#lineNumber45)

Register a new Identity with an Identity Provider

### public registerWithRP() [source](../../../file/src/identity/IdentityModule.js.html#lineNumber52)

In relation with a classical Relying Party: Registration

### public setHypertyIdentity() [source](../../../file/src/identity/IdentityModule.js.html#lineNumber207)

In relation with a Hyperty Instance: Associate identity

### public validateAssertion(assertion: DOMString): [Promise](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise) [source](../../../file/src/identity/IdentityModule.js.html#lineNumber233)

Function to validate an identity assertion generated previously. Returns a promise with the result from the validation.

#### Params:

Name Type Attribute Description assertion DOMString

#### Return:

|  |  |
| --- | --- |
| Promise</s pan> | Promise promise with the result from the validation |

Generated by [ESDoc(0.4.5)](https://esdoc.org) [source](../../../file/src/bus/MessageBus.js.html#lineNumber7)

# MessageBus

#### Extends:

[Bus](../../../class/src/bus/Bus.js~Bus.html) → MessageBus

Message BUS Interface is an extension of the MiniBus It doesn’t support the default ‘\*’ listener, instead it uses the registry.resolve(..)

## Constructor Summary

Public Constructor public

[constructor](../../../class/src/bus/MessageBus.js~MessageBus.html#instance-constructor-constructor)(registry: \*)

## Member Summary

Public Members public get

[pipeline](../../../class/src/bus/MessageBus.js~MessageBus.html#instance-get-pipeline): \*

## Method Summary

Public Methods public

[addForward](../../../class/src/bus/MessageBus.js~MessageBus.html#instance-method-addForward)(from: \*, to: \*): \*

public

[postMessage](../../../class/src/bus/MessageBus.js~MessageBus.html#instance-method-postMessage)(inMsg: \*, responseCallback: \*): \*

## Inherited Summary

From class [Bus](../../../class/src/bus/Bus.js~Bus.html) public

[addListener](../../../class/src/bus/Bus.js~Bus.html#instance-method-addListener)(url: URL, listener: Listener): MsgListener

Register listener to receive message when “msg.to === url”.

public

[addResponseListener](../../../class/src/bus/Bus.js~Bus.html#instance-method-addResponseListener)(url: URL, msgId: [number](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number), responseListener: [Function](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function))

Manually add a response listener.

public

[bind](../../../class/src/bus/Bus.js~Bus.html#instance-method-bind)(outUrl: URL, inUrl: URL, target: [MiniBus](../../../class/src/bus/MiniBus.js~MiniBus.html)): Bound

Helper method to bind listeners (in both directions) into other MiniBus target.

public

[postMessage](../../../class/src/bus/Bus.js~Bus.html#instance-method-postMessage)(msg: Message, responseCallback: [Function](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function)): [number](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number)

Send messages to local listeners, or if not exists to external listeners.

public

[removeAllListenersOf](../../../class/src/bus/Bus.js~Bus.html#instance-method-removeAllListenersOf)(url: URL)

Remove all existent listeners for the URL

public

[removeResponseListener](../../../class/src/bus/Bus.js~Bus.html#instance-method-removeResponseListener)(url: URL, msgId: [number](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number))

Remove the response listener.

## Public Constructors

### public constructor(registry: \*) [source](../../../file/src/bus/MessageBus.js.html#lineNumber19)

#### Override:

[Bus#constructor](../../../class/src/bus/Bus.js~Bus.html#instance-constructor-constructor)

#### Params:

Name Type Attribute Description registry \*

## Public Members

### public get pipeline: \* [source](../../../file/src/bus/MessageBus.js.html#lineNumber29)

## Public Methods

### public addForward(from: \*, to: \*): \* [source](../../../file/src/bus/MessageBus.js.html#lineNumber55)

#### Params:

Name Type Attribute Description from \* to \*

#### Return:

|  |
| --- |
| \* |

### public postMessage(inMsg: \*, responseCallback: \*): \* [source](../../../file/src/bus/MessageBus.js.html#lineNumber31)

#### Override:

[Bus#postMessage](../../../class/src/bus/Bus.js~Bus.html#instance-method-postMessage)

#### Params:

Name Type Attribute Description inMsg \* responseCallback \*

#### Return:

|  |
| --- |
| \* |

Generated by [ESDoc(0.4.5)](https://esdoc.org) [source](../../../file/src/bus/MiniBus.js.html#lineNumber3)

# MiniBus

#### Extends:

[Bus](../../../class/src/bus/Bus.js~Bus.html) → MiniBus

#### Direct Subclass:

[Sandbox](../../../class/src/sandbox/Sandbox.js~Sandbox.html)

## Constructor Summary

Public Constructor public

[constructor](../../../class/src/bus/MiniBus.js~MiniBus.html#instance-constructor-constructor)

## Method Summary

Public Methods public

[postMessage](../../../class/src/bus/MiniBus.js~MiniBus.html#instance-method-postMessage)(inMsg: \*, responseCallback: \*): \*

## Inherited Summary

From class [Bus](../../../class/src/bus/Bus.js~Bus.html) public

[addListener](../../../class/src/bus/Bus.js~Bus.html#instance-method-addListener)(url: URL, listener: Listener): MsgListener

Register listener to receive message when “msg.to === url”.

public

[addResponseListener](../../../class/src/bus/Bus.js~Bus.html#instance-method-addResponseListener)(url: URL, msgId: [number](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number), responseListener: [Function](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function))

Manually add a response listener.

public

[bind](../../../class/src/bus/Bus.js~Bus.html#instance-method-bind)(outUrl: URL, inUrl: URL, target: [MiniBus](../../../class/src/bus/MiniBus.js~MiniBus.html)): Bound

Helper method to bind listeners (in both directions) into other MiniBus target.

public

[postMessage](../../../class/src/bus/Bus.js~Bus.html#instance-method-postMessage)(msg: Message, responseCallback: [Function](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function)): [number](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number)

Send messages to local listeners, or if not exists to external listeners.

public

[removeAllListenersOf](../../../class/src/bus/Bus.js~Bus.html#instance-method-removeAllListenersOf)(url: URL)

Remove all existent listeners for the URL

public

[removeResponseListener](../../../class/src/bus/Bus.js~Bus.html#instance-method-removeResponseListener)(url: URL, msgId: [number](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number))

Remove the response listener.

## Public Constructors

### public constructor [source](../../../file/src/bus/MiniBus.js.html#lineNumber5)

#### Override:

[Bus#constructor](../../../class/src/bus/Bus.js~Bus.html#instance-constructor-constructor)

## Public Methods

### public postMessage(inMsg: \*, responseCallback: \*): \* [source](../../../file/src/bus/MiniBus.js.html#lineNumber9)

#### Override:

[Bus#postMessage](../../../class/src/bus/Bus.js~Bus.html#instance-method-postMessage)

#### Params:

Name Type Attribute Description inMsg \* responseCallback \*

#### Return:

|  |
| --- |
| \* |

Generated by [ESDoc(0.4.5)](https://esdoc.org) [source](../../../file/src/syncher/ObjectAllocation.js.html#lineNumber1)

# ObjectAllocation

## Constructor Summary

Public Constructor public

[constructor](../../../class/src/syncher/ObjectAllocation.js~ObjectAllocation.html#instance-constructor-constructor)(url: URL.URL, bus: [MiniBus](../../../class/src/bus/MiniBus.js~MiniBus.html))

Create an Object Allocation

## Member Summary

Public Members public get

[url](../../../class/src/syncher/ObjectAllocation.js~ObjectAllocation.html#instance-get-url): [string](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String): \*

get the URL value

## Method Summary

Public Methods public

[create](../../../class/src/syncher/ObjectAllocation.js~ObjectAllocation.html#instance-method-create)(domain: Domain, number: [number](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number)): [Promise](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise)<ObjectURL>

Ask for creation of a number of Object addresses, to the domain message node.

## Public Constructors

### public constructor(url: URL.URL, bus: [MiniBus](../../../class/src/bus/MiniBus.js~MiniBus.html)) [source](../../../file/src/syncher/ObjectAllocation.js.html#lineNumber12)

Create an Object Allocation

#### Params:

Name Type Attribute Description url URL.URL url from who is sending the message

bus [MiniBus](../../../class/src/bus/MiniBus.js~MiniBus.html) MiniBus used for address allocation

## Public Members

### public get url: [string](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String): \* [source](../../../file/src/syncher/ObjectAllocation.js.html#lineNumber23)

get the URL value

#### Return:

|  |  |
| --- | --- |
| string</spa n> | The url value; |

## Public Methods

### public create(domain: Domain, number: [number](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number)): [Promise](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise)<ObjectURL> [source](../../../file/src/syncher/ObjectAllocation.js.html#lineNumber31)

Ask for creation of a number of Object addresses, to the domain message node.

#### Params:

Name Type Attribute Description domain Domain Domain of the message node.

number [number](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number) Number of addresses to request

#### Return:

|  |  |
| --- | --- |
| Promise</s pan><ObjectURL&gt ; | A list of ObjectURL’s |

Generated by [ESDoc(0.4.5)](https://esdoc.org) [source](../../../file/src/bus/Pipeline.js.html#lineNumber6)

# Pipeline

## Constructor Summary

Public Constructor public

[constructor](../../../class/src/bus/Pipeline.js~Pipeline.html#instance-constructor-constructor)(\_onFail: \*)

## Method Summary

Public Methods public

[process](../../../class/src/bus/Pipeline.js~Pipeline.html#instance-method-process)(msg: \*, onDeliver: \*)

## Public Constructors

### public constructor(\_onFail: \*) [source](../../../file/src/bus/Pipeline.js.html#lineNumber12)

#### Params:

Name Type Attribute Description \_onFail \*

## Public Methods

### public process(msg: \*, onDeliver: \*) [source](../../../file/src/bus/Pipeline.js.html#lineNumber19)

#### Params:

Name Type Attribute Description msg \* onDeliver \*

Generated by [ESDoc(0.4.5)](https://esdoc.org) [source](../../../file/src/policy/Policy.js.html#lineNumber3)

# PolicyEngine

## Constructor Summary

Public Constructor public

[constructor](../../../class/src/policy/PolicyEngine.js~PolicyEngine.html#instance-constructor-constructor)(identityModule: \*, runtimeRegistry: \*)

## Method Summary

Public Methods public

[addPolicies](../../../class/src/policy/PolicyEngine.js~PolicyEngine.html#instance-method-addPolicies)(key: \*, policies: \*)

public

[authorise](../../../class/src/policy/PolicyEngine.js~PolicyEngine.html#instance-method-authorise)(message: \*): \*

public

[checkPolicies](../../../class/src/policy/PolicyEngine.js~PolicyEngine.html#instance-method-checkPolicies)(message: \*, userID: \*): \*

public

[getApplicablePolicies](../../../class/src/policy/PolicyEngine.js~PolicyEngine.html#instance-method-getApplicablePolicies)(hypertyFrom: \*, hypertyTo: \*, userID: \*): \*

public

[getAuthorisationDecision](../../../class/src/policy/PolicyEngine.js~PolicyEngine.html#instance-method-getAuthorisationDecision)(results: \*): \*

public

[removePolicies](../../../class/src/policy/PolicyEngine.js~PolicyEngine.html#instance-method-removePolicies)(key: \*, policyId: \*)

public

[simulate](../../../class/src/policy/PolicyEngine.js~PolicyEngine.html#instance-method-simulate)(key: \*)

## Public Constructors

### public constructor(identityModule: \*, runtimeRegistry: \*) [source](../../../file/src/policy/PolicyEngine.js.html#lineNumber7)

#### Params:

Name Type Attribute Description identityModule \* runtimeRegistry \*

## Public Methods

### public addPolicies(key: \*, policies: \*) [source](../../../file/src/policy/PolicyEngine.js.html#lineNumber15)

#### Params:

Name Type Attribute Description key \* policies \*

### public authorise(message: \*): \* [source](../../../file/src/policy/PolicyEngine.js.html#lineNumber81)

#### Params:

Name Type Attribute Description message \*

#### Return:

|  |
| --- |
| \* |

### public checkPolicies(message: \*, userID: \*): \* [source](../../../file/src/policy/PolicyEngine.js.html#lineNumber113)

#### Params:

Name Type Attribute Description message \* userID \*

#### Return:

|  |
| --- |
| \* |

### public getApplicablePolicies(hypertyFrom: \*, hypertyTo: \*, userID: \*): \* [source](../../../file/src/policy/PolicyEngine.js.html#lineNumber129)

#### Params:

Name Type Attribute Description hypertyFrom \* hypertyTo \* userID \*

#### Return:

|  |
| --- |
| \* |

### public getAuthorisationDecision(results: \*): \* [source](../../../file/src/policy/PolicyEngine.js.html#lineNumber151)

#### Params:

Name Type Attribute Description results \*

#### Return:

|  |
| --- |
| \* |

### public removePolicies(key: \*, policyId: \*) [source](../../../file/src/policy/PolicyEngine.js.html#lineNumber59)

#### Params:

Name Type Attribute Description key \* policyId \*

### public simulate(key: \*) [source](../../../file/src/policy/PolicyEngine.js.html#lineNumber25)

#### Params:

Name Type Attribute Description key \*

Generated by [ESDoc(0.4.5)](https://esdoc.org) [source](../../../file/src/registry/Registry.js.html#lineNumber12)

# Registry

#### Extends:

[EventEmitter](../../../class/src/utils/EventEmitter.js~EventEmitter.html) → Registry

Runtime Registry Interface

## Constructor Summary

Public Constructor public

[constructor](../../../class/src/registry/Registry.js~Registry.html#instance-constructor-constructor)(msgbus: [MessageBus](../../../class/src/bus/MessageBus.js~MessageBus.html), runtimeURL: HypertyRuntimeURL, appSandbox: AppSandbox, remoteRegistry: DomainURL)

To initialise the Runtime Registry with the RuntimeURL that will be the basis to derive the internal runtime addresses when allocating addresses to internal runtime component.

## Member Summary

Public Members public get

[messageBus](../../../class/src/registry/Registry.js~Registry.html#instance-get-messageBus)(messageBus: [MessageBus](../../../class/src/bus/MessageBus.js~MessageBus.html)): \*

return the messageBus in this Registry

public set

[messageBus](../../../class/src/registry/Registry.js~Registry.html#instance-set-messageBus)(messageBus: [MessageBus](../../../class/src/bus/MessageBus.js~MessageBus.html)): \*

Set the messageBus in this Registry

## Method Summary

Public Methods public

[discoverProtostub](../../../class/src/registry/Registry.js~Registry.html#instance-method-discoverProtostub)(DomainURL: DomainURL): RuntimeURL

To discover protocol stubs available in the runtime for a certain domain.

public

[getAppSandbox](../../../class/src/registry/Registry.js~Registry.html#instance-method-getAppSandbox)(): \*

This function is used to return the sandbox instance where the Application is executing.

public

[getSandbox](../../../class/src/registry/Registry.js~Registry.html#instance-method-getSandbox)(DomainURL: DomainURL): RuntimeSandbox

To discover sandboxes available in the runtime for a certain domain.

public

[getUserHyperty](../../../class/src/registry/Registry.js~Registry.html#instance-method-getUserHyperty)(email: \*): \*

Function to query the Domain registry, with an user email.

public

[onEvent](../../../class/src/registry/Registry.js~Registry.html#instance-method-onEvent)()

To receive status events from components registered in the Registry.

public

[registerHyperty](../../../class/src/registry/Registry.js~Registry.html#instance-method-registerHyperty)(sandbox: [Sandbox](../../../class/src/sandbox/Sandbox.js~Sandbox.html), HypertyCatalogueURL: HypertyCatalogueURL): HypertyURL

To register a new Hyperty in the runtime which returns the HypertyURL allocated to the new Hyperty.

public

[registerPEP](../../../class/src/registry/Registry.js~Registry.html#instance-method-registerPEP)(postMessage: Message.Message, HypertyURL: HypertyURL): HypertyRuntimeURL

To register a new Policy Enforcer in the runtime including as input parameters the function to postMessage, the HypertyURL associated with the PEP, which returns the RuntimeURL allocated to the new Policy Enforcer component.

public

[registerStub](../../../class/src/registry/Registry.js~Registry.html#instance-method-registerStub)(Sandbox: [Sandbox](../../../class/src/sandbox/Sandbox.js~Sandbox.html), DomainURL: DomainURL): RuntimeProtoStubURL

To register a new Protocol Stub in the runtime including as input parameters the function to postMessage, the DomainURL that is connected with the stub, which returns the RuntimeURL allocated to the new ProtocolStub.

public

[resolve](../../../class/src/registry/Registry.js~Registry.html#instance-method-resolve)(url: URL.URL): [Promise](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise)<URL.URL>

To verify if source is valid and to resolve target runtime url address if needed (eg protostub runtime url in case the message is to be dispatched to a remote endpoint).

public

[unregisterHyperty](../../../class/src/registry/Registry.js~Registry.html#instance-method-unregisterHyperty)(HypertyURL: HypertyURL): \*

To unregister a previously registered Hyperty

public

[unregisterPEP](../../../class/src/registry/Registry.js~Registry.html#instance-method-unregisterPEP)(HypertyRuntimeURL: HypertyRuntimeURL): \*

To unregister a previously registered protocol stub

public

[unregisterStub](../../../class/src/registry/Registry.js~Registry.html#instance-method-unregisterStub)(HypertyRuntimeURL: HypertyRuntimeURL): \*

To unregister a previously registered protocol stub

## Inherited Summary

From class [EventEmitter](../../../class/src/utils/EventEmitter.js~EventEmitter.html) public

[addEventListener](../../../class/src/utils/EventEmitter.js~EventEmitter.html#instance-method-addEventListener)(eventType: [string](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String), cb: [Function](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function))

addEventListener listen for an eventType

public

[trigger](../../../class/src/utils/EventEmitter.js~EventEmitter.html#instance-method-trigger)(eventType: [string](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String), params: [object](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object))

Invoke the eventType

## Public Constructors

### public constructor(msgbus: [MessageBus](../../../class/src/bus/MessageBus.js~MessageBus.html), runtimeURL: HypertyRuntimeURL, appSandbox: AppSandbox, remoteRegistry: DomainURL) [source](../../../file/src/registry/Registry.js.html#lineNumber21)

To initialise the Runtime Registry with the RuntimeURL that will be the basis to derive the internal runtime addresses when allocating addresses to internal runtime component. In addition, the Registry domain back-end to be used to remotely register Runtime components, is also passed as input parameter.

#### Params:

Name Type Attribute Description msgbus [MessageBus](../../../class/src/bus/MessageBus.js~MessageBus.html) msgbus

runtimeURL HypertyRuntimeURL runtimeURL

appSandbox AppSandbox appSandbox

remoteRegistry DomainURL remoteRegistry

## Public Members

### public get messageBus(messageBus: [MessageBus](../../../class/src/bus/MessageBus.js~MessageBus.html)): \* [source](../../../file/src/registry/Registry.js.html#lineNumber59)

return the messageBus in this Registry

### public set messageBus(messageBus: [MessageBus](../../../class/src/bus/MessageBus.js~MessageBus.html)): \* [source](../../../file/src/registry/Registry.js.html#lineNumber68)

Set the messageBus in this Registry

## Public Methods

### public discoverProtostub(DomainURL: DomainURL): RuntimeURL [source](../../../file/src/registry/Registry.js.html#lineNumber262)

To discover protocol stubs available in the runtime for a certain domain. If available, it returns the runtime url for the protocol stub that connects to the requested domain. Required by the runtime BUS to route messages to remote servers or peers (do we need something similar for Hyperties?).

#### Params:

Name Type Attribute Description DomainURL DomainURL url

#### Return:

|  |  |
| --- | --- |
| RuntimeURL | RuntimeURL |

### public getAppSandbox(): \* [source](../../../file/src/registry/Registry.js.html#lineNumber83)

This function is used to return the sandbox instance where the Application is executing. It is assumed there is just one App per Runtime instance.

#### Return:

|  |
| --- |
| \* |

### public getSandbox(DomainURL: DomainURL): RuntimeSandbox [source](../../../file/src/registry/Registry.js.html#lineNumber394)

To discover sandboxes available in the runtime for a certain domain. Required by the runtime UA to avoid more than one sandbox for the same domain.

#### Params:

Name Type Attribute Description DomainURL DomainURL url

#### Return:

|  |  |
| --- | --- |
| RuntimeSandbox | RuntimeSandbox |

### public getUserHyperty(email: \*): \* [source](../../../file/src/registry/Registry.js.html#lineNumber91)

Function to query the Domain registry, with an user email.

#### Params:

Name Type Attribute Description email \*

#### Return:

|  |
| --- |
| \* |

### public onEvent() [source](../../../file/src/registry/Registry.js.html#lineNumber385)

To receive status events from components registered in the Registry.

#### Params:

Name Type Attribute Description Message.Message Message.Message event

### public registerHyperty(sandbox: [Sandbox](../../../class/src/sandbox/Sandbox.js~Sandbox.html), HypertyCatalogueURL: HypertyCatalogueURL): HypertyURL [source](../../../file/src/registry/Registry.js.html#lineNumber148)

To register a new Hyperty in the runtime which returns the HypertyURL allocated to the new Hyperty.

#### Params:

Name Type Attribute Description sandbox [Sandbox](../../../class/src/sandbox/Sandbox.js~Sandbox.html) sandbox

HypertyCatalogueURL HypertyCatalogueURL descriptor

#### Return:

|  |  |
| --- | --- |
| HypertyURL | HypertyURL |

### public registerPEP(postMessage: Message.Message, HypertyURL: HypertyURL): HypertyRuntimeURL [source](../../../file/src/registry/Registry.js.html#lineNumber350)

To register a new Policy Enforcer in the runtime including as input parameters the function to postMessage, the HypertyURL associated with the PEP, which returns the RuntimeURL allocated to the new Policy Enforcer component.

#### Params:

Name Type Attribute Description postMessage Message.Message postMessage

HypertyURL HypertyURL hyperty

#### Return:

|  |  |
| --- | --- |
| HypertyRuntimeURL | HypertyRuntimeURL |

### public registerStub(Sandbox: [Sandbox](../../../class/src/sandbox/Sandbox.js~Sandbox.html), DomainURL: DomainURL): RuntimeProtoStubURL [source](../../../file/src/registry/Registry.js.html#lineNumber285)

To register a new Protocol Stub in the runtime including as input parameters the function to postMessage, the DomainURL that is connected with the stub, which returns the RuntimeURL allocated to the new ProtocolStub.

#### Params:

Name Type Attribute Description Sandbox [Sandbox](../../../class/src/sandbox/Sandbox.js~Sandbox.html) DomainURL DomainURL service provider domain

#### Return:

|  |
| --- |
| RuntimeProtoStubURL |

### public resolve(url: URL.URL): [Promise](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise)<URL.URL> [source](../../../file/src/registry/Registry.js.html#lineNumber439)

To verify if source is valid and to resolve target runtime url address if needed (eg protostub runtime url in case the message is to be dispatched to a remote endpoint).

#### Params:

Name Type Attribute Description url URL.URL url

#### Return:

|  |  |
| --- | --- |
| Promise</s pan><URL.URL> | Promise <URL.URL> |

### public unregisterHyperty(HypertyURL: HypertyURL): \* [source](../../../file/src/registry/Registry.js.html#lineNumber229)

To unregister a previously registered Hyperty

#### Params:

Name Type Attribute Description HypertyURL HypertyURL url url

#### Return:

|  |
| --- |
| \* |

### public unregisterPEP(HypertyRuntimeURL: HypertyRuntimeURL): \* [source](../../../file/src/registry/Registry.js.html#lineNumber365)

To unregister a previously registered protocol stub

#### Params:

Name Type Attribute Description HypertyRuntimeURL HypertyRuntimeURL HypertyRuntimeURL

#### Return:

|  |
| --- |
| \* |

### public unregisterStub(HypertyRuntimeURL: HypertyRuntimeURL): \* [source](../../../file/src/registry/Registry.js.html#lineNumber326)

To unregister a previously registered protocol stub

#### Params:

Name Type Attribute Description HypertyRuntimeURL HypertyRuntimeURL hypertyRuntimeURL

#### Return:

|  |
| --- |
| \* |

Generated by [ESDoc(0.4.5)](https://esdoc.org) [source](../../../file/src/registry/RegistryDataModel.js.html#lineNumber5)

# RuntimeCatalogue

## Constructor Summary

Public Constructor public

[constructor](../../../class/src/runtime/RuntimeCatalogue.js~RuntimeCatalogue.html#instance-constructor-constructor)(nodeHttp: \*, nodeHttps: \*)

## Member Summary

Public Members public set

[runtimeURL](../../../class/src/runtime/RuntimeCatalogue.js~RuntimeCatalogue.html#instance-set-runtimeURL): \*

public get

[runtimeURL](../../../class/src/runtime/RuntimeCatalogue.js~RuntimeCatalogue.html#instance-get-runtimeURL): \*

## Method Summary

Public Methods public

[getDataSchemaDescriptor](../../../class/src/runtime/RuntimeCatalogue.js~RuntimeCatalogue.html#instance-method-getDataSchemaDescriptor)(dataSchemaURL: \*): [Promise](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise)

Get DataSchemaDescriptor

public

[getHypertyDescriptor](../../../class/src/runtime/RuntimeCatalogue.js~RuntimeCatalogue.html#instance-method-getHypertyDescriptor)(hypertyURL: \*): [Promise](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise)

Get HypertyDescriptor

public

[getHypertyRuntimeURL](../../../class/src/runtime/RuntimeCatalogue.js~RuntimeCatalogue.html#instance-method-getHypertyRuntimeURL)(): \*

Get hypertyRuntimeURL

public

[getRuntimeDescriptor](../../../class/src/runtime/RuntimeCatalogue.js~RuntimeCatalogue.html#instance-method-getRuntimeDescriptor)(runtimeURL: \*): [Promise](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise)

Get RuntimeDescriptor

public

[getSourceCodeFromDescriptor](../../../class/src/runtime/RuntimeCatalogue.js~RuntimeCatalogue.html#instance-method-getSourceCodeFromDescriptor)(descriptor: \*): [Promise](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise)

Returns the sourceCode of a given descriptor

public

[getSourcePackageFromURL](../../../class/src/runtime/RuntimeCatalogue.js~RuntimeCatalogue.html#instance-method-getSourcePackageFromURL)(sourcePackageURL: \*): [Promise](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise)

Get source Package from a URL

public

[getStubDescriptor](../../../class/src/runtime/RuntimeCatalogue.js~RuntimeCatalogue.html#instance-method-getStubDescriptor)(stubURL: \*): [Promise](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise)

Get StubDescriptor

## Public Constructors

### public constructor(nodeHttp: \*, nodeHttps: \*) [source](../../../file/src/runtime/RuntimeCatalogue.js.html#lineNumber7)

#### Params:

Name Type Attribute Description nodeHttp \* nodeHttps \*

## Public Members

### public set runtimeURL: \* [source](../../../file/src/runtime/RuntimeCatalogue.js.html#lineNumber15)

### public get runtimeURL: \* [source](../../../file/src/runtime/RuntimeCatalogue.js.html#lineNumber20)

## Public Methods

### public getDataSchemaDescriptor(dataSchemaURL: \*): [Promise](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise) [source](../../../file/src/runtime/RuntimeCatalogue.js.html#lineNumber251)

Get DataSchemaDescriptor

#### Params:

Name Type Attribute Description dataSchemaURL \* e.g. mydomain.com/.well-known/dataschema/MyDataSchema

#### Return:

|  |
| --- |
| [Promise](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise) |

### public getHypertyDescriptor(hypertyURL: \*): [Promise](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise) [source](../../../file/src/runtime/RuntimeCatalogue.js.html#lineNumber145)

Get HypertyDescriptor

#### Params:

Name Type Attribute Description hypertyURL \* e.g. mydomain.com/.well-known/hyperty/MyHyperty

#### Return:

|  |
| --- |
| [Promise](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise) |

### public getHypertyRuntimeURL(): \* [source](../../../file/src/runtime/RuntimeCatalogue.js.html#lineNumber28)

Get hypertyRuntimeURL

#### Return:

|  |
| --- |
| \* |

### public getRuntimeDescriptor(runtimeURL: \*): [Promise](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise) [source](../../../file/src/runtime/RuntimeCatalogue.js.html#lineNumber193)

Get RuntimeDescriptor

#### Params:

Name Type Attribute Description runtimeURL \* e.g. mydomain.com/.well-known/runtime/MyRuntime

#### Return:

|  |
| --- |
| [Promise](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise) |

### public getSourceCodeFromDescriptor(descriptor: \*): [Promise](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise) [source](../../../file/src/runtime/RuntimeCatalogue.js.html#lineNumber391)

Returns the sourceCode of a given descriptor

#### Params:

Name Type Attribute Description descriptor \* Catalogue Object that was retrieved using e.g. getHypertyDescriptor()

#### Return:

|  |
| --- |
| [Promise](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise) |

### public getSourcePackageFromURL(sourcePackageURL: \*): [Promise](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise) [source](../../../file/src/runtime/RuntimeCatalogue.js.html#lineNumber298)

Get source Package from a URL

#### Params:

Name Type Attribute Description sourcePackageURL \* e.g. mydomain.com/.well-known/hyperty/MyHyperty/sourcePackage

#### Return:

|  |
| --- |
| [Promise](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise) |

### public getStubDescriptor(stubURL: \*): [Promise](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise) [source](../../../file/src/runtime/RuntimeCatalogue.js.html#lineNumber327)

Get StubDescriptor

#### Params:

Name Type Attribute Description stubURL \* e.g. mydomain.com/.well-known/protostub/MyProtostub

#### Return:

|  |
| --- |
| [Promise](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise) |

Generated by [ESDoc(0.4.5)](https://esdoc.org) data-ice=“source”>| [source](../../../file/src/runtime/RuntimeUA.js.html#lineNumber28)

# RuntimeUA

Runtime User Agent Interface will process all the dependecies of the core runtime;

## Constructor Summary

Public Constructor public

[constructor](../../../class/src/runtime/RuntimeUA.js~RuntimeUA.html#instance-constructor-constructor)(sandboxFactory: sandboxFactory, domainURL: domain)

Create a new instance of Runtime User Agent

## Method Summary

Public Methods public

[checkForUpdate](../../../class/src/runtime/RuntimeUA.js~RuntimeUA.html#instance-method-checkForUpdate)(url: CatalogueURL)

Used to check for updates about components handled in the Catalogue including protocol stubs and Hyperties.

public

[discoverHiperty](../../../class/src/runtime/RuntimeUA.js~RuntimeUA.html#instance-method-discoverHiperty)(descriptor: CatalogueDataObject.HypertyDescriptor)

Accomodate interoperability in H2H and proto on the fly for newly discovered devices in M2M

public

[loadHyperty](../../../class/src/runtime/RuntimeUA.js~RuntimeUA.html#instance-method-loadHyperty)(hyperty: URL.HypertyCatalogueURL): \*

Deploy Hyperty from Catalogue URL

public

[loadStub](../../../class/src/runtime/RuntimeUA.js~RuntimeUA.html#instance-method-loadStub)(domain: URL.URL): \*

Deploy Stub from Catalogue URL or domain url

public

[registerHyperty](../../../class/src/runtime/RuntimeUA.js~RuntimeUA.html#instance-method-registerHyperty)(Object: [Object](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object), descriptor: URL.HypertyCatalogueURL)

Register Hyperty deployed by the App that is passed as input parameter.

## Public Constructors

### public constructor(sandboxFactory: sandboxFactory, domainURL: domain) [source](../../../file/src/runtime/RuntimeUA.js.html#lineNumber35)

Create a new instance of Runtime User Agent

#### Params:

Name Type Attribute Description sandboxFactory sandboxFactory Specific implementation for the environment where the core runtime will run;

domainURL domain specify the domain base for the runtime;

## Public Methods

### public checkForUpdate(url: CatalogueURL) [source](../../../file/src/runtime/RuntimeUA.js.html#lineNumber460)

Used to check for updates about components handled in the Catalogue including protocol stubs and Hyperties. check relationship with lifecycle management provided by Service Workers

#### Params:

Name Type Attribute Description url CatalogueURL url

### public discoverHiperty(descriptor: CatalogueDataObject.HypertyDescriptor) [source](../../../file/src/runtime/RuntimeUA.js.html#lineNumber119)

Accomodate interoperability in H2H and proto on the fly for newly discovered devices in M2M

#### Params:

Name Type Attribute Description descriptor CatalogueDataObject.HypertyDescriptor descriptor

### public loadHyperty(hyperty: URL.HypertyCatalogueURL): \* [source](../../../file/src/runtime/RuntimeUA.js.html#lineNumber136)

Deploy Hyperty from Catalogue URL

#### Params:

Name Type Attribute Description hyperty URL.HypertyCatalogueURL hypertyDescriptor url;

#### Return:

|  |
| --- |
| \* |

### public loadStub(domain: URL.URL): \* [source](../../../file/src/runtime/RuntimeUA.js.html#lineNumber306)

Deploy Stub from Catalogue URL or domain url

#### Params:

Name Type Attribute Description domain URL.URL domain

#### Return:

|  |
| --- |
| \* |

### public registerHyperty(Object: [Object](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object), descriptor: URL.HypertyCatalogueURL) [source](../../../file/src/runtime/RuntimeUA.js.html#lineNumber128)

Register Hyperty deployed by the App that is passed as input parameter. To be used when App and Hyperties are from the same domain otherwise the RuntimeUA will raise an exception and the App has to use the loadHyperty(..) function.

#### Params:

Name Type Attribute Description Object [Object](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object) hypertyInstance

descriptor URL.HypertyCatalogueURL descriptor

Generated by [ESDoc(0.4.5)](https://esdoc.org) [source](../../../file/src/sandbox/Sandbox.js.html#lineNumber9)

# Sandbox

#### Extends:

[Bus](../../../class/src/bus/Bus.js~Bus.html) → [MiniBus](../../../class/src/bus/MiniBus.js~MiniBus.html) → Sandbox

## Constructor Summary

Public Constructor public

[constructor](../../../class/src/sandbox/Sandbox.js~Sandbox.html#instance-constructor-constructor)

## Method Summary

Public Methods public

[deployComponent](../../../class/src/sandbox/Sandbox.js~Sandbox.html#instance-method-deployComponent)(componentSourceCode: [string](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String), componentURL: URL, configuration: Config): [Promise](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise)<[string](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String)>

Deploy an instance of the component into the sandbox.

public

[removeComponent](../../../class/src/sandbox/Sandbox.js~Sandbox.html#instance-method-removeComponent)(componentURL: URL): [Promise](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise)<[string](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String)>

Remove the instance of a previously deployed component.

## Inherited Summary

From class [Bus](../../../class/src/bus/Bus.js~Bus.html) public

[addListener](../../../class/src/bus/Bus.js~Bus.html#instance-method-addListener)(url: URL, listener: Listener): MsgListener

Register listener to receive message when “msg.to === url”.

public

[addResponseListener](../../../class/src/bus/Bus.js~Bus.html#instance-method-addResponseListener)(url: URL, msgId: [number](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number), responseListener: [Function](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function))

Manually add a response listener.

public

[bind](../../../class/src/bus/Bus.js~Bus.html#instance-method-bind)(outUrl: URL, inUrl: URL, target: [MiniBus](../../../class/src/bus/MiniBus.js~MiniBus.html)): Bound

Helper method to bind listeners (in both directions) into other MiniBus target.

public

[postMessage](../../../class/src/bus/Bus.js~Bus.html#instance-method-postMessage)(msg: Message, responseCallback: [Function](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function)): [number](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number)

Send messages to local listeners, or if not exists to external listeners.

public

[removeAllListenersOf](../../../class/src/bus/Bus.js~Bus.html#instance-method-removeAllListenersOf)(url: URL)

Remove all existent listeners for the URL

public

[removeResponseListener](../../../class/src/bus/Bus.js~Bus.html#instance-method-removeResponseListener)(url: URL, msgId: [number](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number))

Remove the response listener.

From class [MiniBus](../../../class/src/bus/MiniBus.js~MiniBus.html) public

[postMessage](../../../class/src/bus/MiniBus.js~MiniBus.html#instance-method-postMessage)(inMsg: \*, responseCallback: \*): \*

## Public Constructors

### public constructor [source](../../../file/src/sandbox/Sandbox.js.html#lineNumber11)

#### Override:

[MiniBus#constructor](../../../class/src/bus/MiniBus.js~MiniBus.html#instance-constructor-constructor)

## Public Methods

### public deployComponent(componentSourceCode: [string](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String), componentURL: URL, configuration: Config): [Promise](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise)<[string](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String)> [source](../../../file/src/sandbox/Sandbox.js.html#lineNumber29)

Deploy an instance of the component into the sandbox.

#### Params:

Name Type Attribute Description componentSourceCode [string](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String) Component source code (Hyperty, ProtoStub, etc)

componentURL URL Hyperty, ProtoStub, or any other component address.

configuration Config Config parameters of the component

#### Return:

|  |  |
| --- | --- |
| Promise</s pan><string> | return deployed if successful, or any other string with an error |

### public removeComponent(componentURL: URL): [Promise](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise)<[string](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String)> [source](../../../file/src/sandbox/Sandbox.js.html#lineNumber59)

Remove the instance of a previously deployed component.

#### Params:

Name Type Attribute Description componentURL URL Hyperty, ProtoStub, or any other component address.

#### Return:

|  |  |
| --- | --- |
| Promise</s pan><string> | return undeployed if successful, or any other string with an error |

Generated by [ESDoc(0.4.5)](https://esdoc.org) [source](../../../file/src/sandbox/SandboxRegistry.js.html#lineNumber8)

# SyncherManager

## Constructor Summary

Public Constructor public

[constructor](../../../class/src/syncher/SyncherManager.js~SyncherManager.html#instance-constructor-constructor)(runtimeURL: \*, bus: \*, registry: \*, catalog: \*, allocator: \*)

## Member Summary

Public Members public get

[url](../../../class/src/syncher/SyncherManager.js~SyncherManager.html#instance-get-url): \*

## Public Constructors

### public constructor(runtimeURL: \*, bus: \*, registry: \*, catalog: \*, allocator: \*) [source](../../../file/src/syncher/SyncherManager.js.html#lineNumber18)

#### Params:

Name Type Attribute Description runtimeURL \* bus \* registry \* catalog \* allocator \*

## Public Members

### public get url: \* [source](../../../file/src/syncher/SyncherManager.js.html#lineNumber50)

Generated by [ESDoc(0.4.5)](https://esdoc.org)