Contents Include:
Resource Oriented Primer
Standard Modules
Identifier Grammars
NetKernel Foundation API

Quick Reference to
NetKernel v5 Resource Oriented Platform

About NetKernel

NetKernel is a Resource Oriented Computing Platform developed by 1060 Research. It is based upon the Java Standard Edition platform but incorporates many standards and implementations from elsewhere. It is distributed in two flavours, Standard Edition which is licensed under an open source 1060 Public License and an Enterprise Edition which is commercially licensed. Enterprise Edition has features to support development and deployment of high performant scalable enterprise systems. Use cases of NetKernel range from application server, rich client, (or combinations of both), with HTTP transports and many others through to very small footprint standalone applications and embedded within J2EE containers.

What makes NetKernel different

NetKernel is based upon the Resource Oriented Computing (ROC) concept rather than the Object Oriented Programming (OOP) concept used by much of today's software. Research initiated at Hewlett Packard Labs showed how this unique approach which combines many of the concepts from REST[1] and UNIX, has many compelling consequences for software systems.

Resource Oriented Primer

ROC is a software paradigm which constrains software more tightly than OOP by providing stronger encapsulation and richer modularity. It decomposes software into endpoints within address spaces which are accessed via request-response pairs transfering state with representations.

Resource Oriented Concepts
Resource – a modelled abstract entity with state. Resources may model physical entities, software entities, data or abstract derivative concepts.
Endpoint – a software service which provides access to read or modify the state of one or more resources.
Representation – an immutable data structure which can be sent to an endpoint to update the resource's state or requested from an endpoint to gain a snapshot of a resource's state.
Request – a message issued by a client to act upon a resource.
Identifier – a character string used to identify resources.
Verb – requests are constrained to one of a small set of well defined actions with well defined semantics, including: SOURCE, SINK, EXISTS, DELETE, NEW.
Resolution – the process of determining the endpoint for a given request.
Space – a container which aggregates endpoints and hence resources and provides a resolution mechanism.
Response – returned from an endpoint after processing a request.
Accessor – the most common type of endpoint, the accessor provides access to a resource's state. Also typically embodies transformation functions and language runtimes.
Transreptor – a specialised endpoint which will convert information in one representation to the same information in another representational form. Usually used to adapt representations between an endpoint and it's client.
Transport – a specialised endpoint which receives events from outside the resource oriented system and adapts them to a request. Transports may return responses too.
Overlay – a specialised endpoint which wraps a space and issues sub-requests into it. This provides the basis for patterns such as importing spaces into other spaces.
Request Scope – an ordered list of spaces within a request which are used to determine resolution.

NetKernel's ROC Implementation

NetKernel provides an embodiment of ROC. Here are some of the pertinent characteristics:

NetKernel Concepts
Microkernel – NetKernel's implementation of ROC uses a microkernel which is responsible for the lifecycle of modules, spaces and their contained endpoints as well as mediating requests.
Asynchronous Requests – Both request issuing and request handling within endpoints is asynchronous. i.e. requests can be issued and their responses can be retrieved later (possibly on a different thread.)
Caching – Both request resolutions and response representations will be cached during execution with minimal configuration. Caching has a rich dependency based approach which ensures stale items are never served.
Module – NetKernel has a modular architecture. Modules contain one or more ROC spaces.
NetKernel Foundation API – NKF provides a common API for endpoints which abstracts the low-level kernel APIs and provides a uniform high level interface.

Downloading and Installation

To download NetKernel Enterprise Edition visit https://cs.1060research.com/csp/download/. Create a portal account if you don't already have one and download a fully featured evaluation version with a 3 month license.

1 Roy Fielding formalised the architecture of the WWW in his doctoral thesis. http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

To download NetKernel Standard Edition visit http:// download.netkernel.org/ and select the latest stable version from a mirror close to you.

The download works across the supported platforms, Windows 2000, Windows XP, Windows Vista, Windows Server 2003, Apple Mac OS X, Linux (Redhat, Suse, Debian, Ubuntu) and Solaris. For either version you will need to have installed a Java virtual machine version 1.5 or greater. Supported platforms are Sun 1.5.x and Sun 1.6.x but should work fine on the equivalent versions from other vendors. Visit *http://www.java.com/getjava/* for downloads.

Running without installation

Once you got the download and ensured Java is installed, open up a terminal window, change directory to your download directory and type:

```
java -jar 1060-NetKernel-EE-x.x.x.jar
```

Like a Linux Live-CD, NetKernel boots directly from the Jar and is fully usable. When NetKernel has started, point your web browser at the NetKernel Admin Panel:

http://localhost:1060

You can explore the NetKernel tools, documentation and tutorials here without installation.

Installation

To start developing on NetKernel you will need to install it onto your filesystem. The Install tab on the Admin Panel provides a self-installer. Follow the prompts to install.

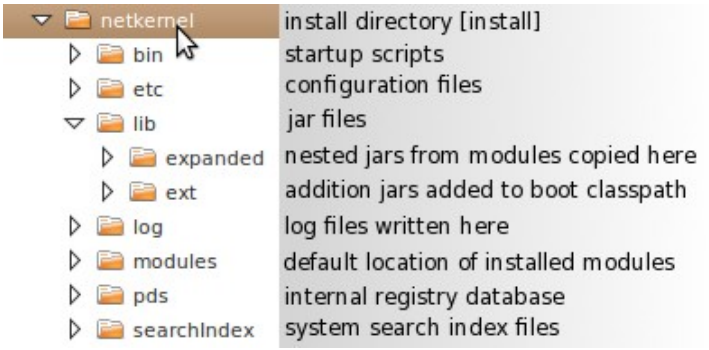

Figure 1: NetKernel installation directory structure

Once installed go to the install directory (in the documentation this is given the shorthand [install]) and type the following at the command-line:

bin/netkernel.sh [Linux/Unix]
bin\netkernel.bat [Windows]

Standard Module

NetKernel supports a pluggable module-implementation architecture. Modules are used to deploy functionality onto a NetKernel instance. By far the most common implementation is the Standard Module.

Standard Modules may be either a directory on the filesystem or a Java jar archive. In each case, the root directory contains a file *module.xml* which defines the metadata about the module and it's static structure. The *module.xml* must be well formed XML. Here is an example of the structure, the important tags are explained below:

```
<module version="2.0">
  <meta>
    <identity>
      <uri>urn:org:myorg:myproject</uri>
      <version>1.0.0</version>
    </identity>
    <info>
      <name>My Project</name>
      <description>My First NetKernel Project</description>
      <icon>res:/org/myorg/myproject/icon.png</icon>
    </info>
  </meta>
  <system>
    <dynamic/>
    <classloader>
      <exports>
        <match>org.myorg.myproject.*</match>
      </exports>
    </classloader>
  </system>
</module>
```

Standard Module Tags	
uri	A globally unique identifier for the module. It is recommended you use URN[2]s.
version	A version specification which must be three integers separated by a dot character.
name	A concise name for the module.
description	A longer description of the module.
icon	An optional identifier to a 48x48 pixel PNG icon. This identifier must be resolvable by the first public space within the module.
dynamic	If this tag is present the module is polled for changes in module.xml and any contained Java classes and will redeploy if changed.
match	By default each module encapsulates it's classloader. If this tag is present then classes matching the regular expression will be exported into importing spaces. Multiple matches can be specified.

Each Standard Module contains a Java classloader which is used to load endpoints and representations. The classloader resolves classes from the following locations in order:

1. The root directory of the module.
2. Each jar file placed in the **/lib/** directory in unspecified order.
3. The *exported* classes from any *imported* spaces in unspecified order.
4. From the boot classloader which contains the kernel and core libraries.
5. From the Java core libraries.

2 http://en.wikipedia.org/wiki/Uniform_Resource_Name

Defining Spaces

One or more spaces can be defined within a standard module. *Root spaces* are top-level spaces that can be imported into other spaces. Rootspaces are defined with a **<rootspace>** tag and have the following attributes:

Rootspace Attributes	
public	Determines if the space is available outside the module. May have the value “true” or “false”. By default rootspaces are public.
name	An optional name for the space. The name is used by tools. The name defaults to the URI of the space.
uri	An optional identifier for the space. The URI is used to identify the space when it is imported. By default the space's URI is generated from the module's URI. The first space has the same URI as the module. Subsequent spaces have the default URI of the module URI appended with “:2”, then “:3”, etc...
private-filter	The optional private-filter tag determines if the space is wrapped with a filter to hide all private endpoints to importing spaces. By default a private filter is used if any contained endpoints are marked as private.

Space Elements

The following elements can be defined within a space:

<accessor>
Define a resource accessor endpoint implemented with Java code.

tags

id	(optional) A unique logical identifier for the accessor. If missing one will be auto-generated.
name	(optional) A name for the accessor used by tools.
description	(optional) A description for the accessor used by tools.
grammar	See the identifier grammars section.
private	If present this tag indicates that the Accessor should not be made available outside this module.
class	The implementation class must be a Java class defined within the classloader for the containing module. The class must extend:

```
org.netkernel.module.standard.endpoint.StandardAccessorImpl
```

override methods

```
// handle SOURCE requests
void onSource(INKFRequestContext context) throws Exception;
// handle SINK requests
void onSink(INKFRequestContext context) throws Exception;
// handle EXISTS requests
void onExists(INKFRequestContext context) throws Exception;
// handle DELETE requests
void onDelete(INKFRequestContext context) throws Exception;
// handle EXISTS requests
void onExists(INKFRequestContext context) throws Exception;
// handle SOURCE, SINK, EXISTS, DELETE, EXISTS requests
void onRequest(INKFRequestContext context) throws Exception;
```

<transport>
Define a transport endpoint to receive external events.

tags

id, name, description	See accessor.
class	The implementation class must be a Java class defined within the classloader for the containing module. The class must extend:

```
org.netkernel.module.standard.endpoint.StandardTransportImpl
```

override methods

```
// start thread or library that will receive external events
void postCommission(INKFRequestContext context) throws Exception;
// stop thread or library
void preDecommission(INKFRequestContext context) throws Exception;
```

<transreptor>
Define a transreptor endpoint to convert resource representations from one form to another.

tags

id, name, description	See accessor.
class	The implementation class must be a Java class defined within the classloader for the containing module. The class must extend:

```
org.netkernel.module.standard.endpoint.StandardTransreptorImpl
```

override methods

```
//constructor should call these methods to specify representations
void declareFromRepresentation(Class representationClass);
void declareToRepresentation(Class representationClass);
// handle TRANSREPT requests
void onTransrept(INKFRequestContext context) throws Exception;
```

<import>
Import another rootspace.

tags

uri	URI of the space to import
private	If present this tag indicates that the imported address space should not be made available outside this module.

<fileset> Expose static resources from the module's directory as resources with the res: scheme	
tags	
id, name, description ,private	See accessor space element description.
glob	Glob expression of resources to expose, e.g. mydir/* for all resources in the sub-directory “mydir”.
regex	Regular expression of resource identifiers to expose, e.g. res:/mydir/.* for all resources in the sub-directory “mydir”. regex and glob are mutually exclusive.
rewrite	Rewrite the identifier so that static resources can be located in an alternate location. For use the the regex tag, the rewrite tag contains an expression that substitutes the captured groups in the regex by %1, %2 etc.
mutable	If present and the jar is expanded resources will support SINK verb to update resource.

<mapper> An overlay to wrap a nested space and map it's resources into the host address space.	
tags	
config	Mapper configuration document described below.
space	A nested space which may contain any space elements.

The mapper configuration is an XML document which consists of a **config** root tag containing one or more **endpoint** tags which define logical endpoints created by mapping grammars to endpoints contained within the nested space. Endpoints may have the following configuration tags:

Mapper endpoint tags	
id, name, description	See accessor space element description.
grammar	See the identifier grammars section.
request	The request to issue into the nested space. See declarative request section.
header	Set header on returned response.
verbs	(Optional) comma separated list of supported verbs. Defaults to verbs of underlying endpoint.
private	Endpoint can only be resolved from inside the nested space. (by sub-requests)
protected	Endpoint is only visible inside this module.

<pluggable-overlay> Wrap a nested space and pass requests into space through a pre-process accessor and/or responses through a post-process accessor.	
tags	
preProcess	Optional declarative request for pre-process service. “arg:request” can be used to reference inbound request to pass to service.
postProcess	Optional declarative request for post-process service. “arg:request” and “arg:response” can be used to reference inbound request and outbound response respectively.
space	A nested space which may contain any space elements.

<branch-merge> An overlay to wrap a nested space, define channels and pass each channel through a chain of handler overlays.	
tags	
config	See main documentation for details.
space	A nested space which may contain any space elements.

Identifier Grammars

Identifiers are used by a number of technologies as a uniform way of specifying the identifier syntax of endpoints. Grammars fulfil three functions: construction of request identifiers by clients to an endpoint, resolution matching of request identifiers to endpoints and extraction of arguments from identifiers by endpoints.

Identifier grammars are defined using an XML schema with the following tags:

Grammar tags	
grammar	The root tag of all grammar declarations.
regex	Defines a sequence of characters defined by either a custom regular expression as text contents, or with a type attribute defining a preset expression: **uri**, **active-escaped-uri, active-escaped-uri-loose, relative-path, relative-directory-path, integer, float, nmtoken, alphanum, anything.**
group	Defines a group of nested tags. Group supports the following attributes: – **name** – the group will be captured as an argument with this name. – **max** - maximum repetition of this group (defaults to “1”, use “*” for any number) – **min** - minimum repetition of this group (defaults to “1”) – **encoding** - optional encoding attribute specifies how arguments are encoded when identifiers are constructed and decoded when extracted. By default no encoding is performed. Two encodings are implemented: – **active** - escapes unsafe characters inside active identifiers. – **url-query** – implements standard URL encoding as per *java.net.URLDecoder*
interleave	Defines a group where nested groups can be specified in any order.

choice	Defines a group where one and only one of the nested groups can be specified.
“text”	Any text placed within the grammar and outside a regex tag will be treated literally.

```
<grammar>http://
  <group name=”hostname”>
    <regex type=”nmtoken”/>
  </group>
  <group name=”path”>/
    <regex>.*</regex>
  </group>
</grammar>
```

Active Identifier Grammars match active URIs. They are a special class of Identifier Grammars with the useful ability to express function invocation. They are the most common grammars, used for services like transform or data endpoints. A compact schema is used with the following tags:

Active Grammar tags	
active	Indicate Active Grammar syntax.
identifier	Defines the starting character sequence of the identifier.
argument	Defines an argument. Arguments are always encoded as “active”. Argument supports the following attributes: – **name** – the name of the argument – **max** - maximum repetition of this argument (defaults to “1”, use “*” for any number) – **min** - minimum repetition of this group (defaults to “1”)
varargs	Allows any number of additional arguments.

```
<grammar>
  <active>
    <identifier>active:xslt</identifier>
    <argument name=”stylesheet”/>
    <argument name=”operand”/>
    <varargs/>
  </active>
</grammar>
```

Declarative Request

Declarative requests are used by many technologies (mapper, XUnit, XRL, pluggable-overlay, etc.) in which requests must be constructed based on configuration. Declarative requests are defined using an XML schema with the following tags:

Declarative Request tags	
request	Root tag of request syntax.
verb	Specify one the verbs: SOURCE, SINK, EXISTS, DELETE, NEW, TRANSREPT, META. Defaults to SOURCE.
identifier	The identifier can operate in three modes: 1) Specify complete identifier and don't use arguments. 2) Specify starting character sequence of an active identifier. The request will be built with an active identifier syntax. 3) Use meta: scheme and specify an endpoint id. The request will be built according to the grammar of the endpoint.
argument	Adds an argument to the request. The request tag has a mandatory **name** attribute. An option method tag may have the values of **value**, **data-uri**, **as-string**. Argument tags may either contain a string/identifier contents or a single **literal** tag.
literal	Defines a representation that will be passed by-value on the request. Literal requires a type attribute which must have on of the following values: **string**, **byte**, **short**, **integer**, **long**, **float**, **double**, **boolean**, **char**, **xml**, **hds** or a fully qualified java class for example *java.net.URI*. Constructor argument for java classes are defined with nested literal tags.
representation	Specify required response representation. This must be a fully qualified java class or interface.

Example:

```
<request>
  <verb>SOURCE</verb>
  <identifier>active:xslt</identifier>
  <argument name=”stylesheet”>res:/mystylesheet.xsl</argument>
  <argument name=”operand”>
    <literal type=”xml”>
      <doc/>
    </literal>
  </argument>
  <representation>org.w3c.dom.Document</representation>
</request>
```

Requests

NetKernel supports a fixed set of request verbs. Each verb has well defined semantics. Endpoints may implement one or more of these verbs. Any request may return back a java.lang.Throwable (exception) if an error happens whilst processing.

Request Verbs	
SOURCE	Use 1) When resource embodies some state or a transformation of state; request a representation of the current state. Should not modify resource. The requested representation may or may not be returned by the resolved endpoint. If not then NetKernel will sequence a TRANSREPT request afterwards. Use 2) When embodies a service; invoke the service. This may possibly cause a state change or have side-effects and expiration of response should indicate this.
SINK	Update the state of a resource with a representation passed as the primary argument. A null response is returned.
EXISTS	Test for the existence of a resource. Will return a java.lang.Boolean TRUE if and only if the request can be resolved, the endpoint implements the

	EXISTS verb and the existence of the resource can be confirmed.
DELETE	Delete the resource. Will return TRUE if resource was successfully deleted.
NEW	Create a new resource seeded with the representation passed as the primary argument. The response is the identifier of the created resource.
TRANSREPT	Convert information in the primary argument to the same information in another representational form specified by the requested representation field on the request.
META	Request the meta data from an endpoint or space. Usually this request is used only by tools or infrastructural endpoints.

These optional standard request headers tweak the behaviour of request handling. Additional application specific headers can be set.

Standard Request Headers	
liveness	Set period (milliseconds) before any liveness tests should consider request non-live.
forget-dependencies	Declare that request should not have its dependencies tracked because it is likely to have too many and that may cause the system to run out of memory tracking them. By not tracking dependencies a response will always be expired. Existence of header causes dependencies to be forgot.
exclude-dependencies	Declare that the response from this request will be excluded from effecting it's parent request. This is particularly useful to stop requests with always expired responses from stopping cacheability.
priority	Set the request priority as an integer.
no-cache	Declare that response for this request should not be retrieved from cache or cached. Usually it is for the endpoint providing the response to determine if the representation should be cached. In some rare situations it is useful for the client to force no caching. Existence of header inhibits caching.

The following optional standard **response headers** tweak the behaviour of response handling. Again, additional application specific headers can be set.

Standard Response Headers	
mime	Multipurpose Internet Mail Extensions (MIME) type meta-data of representation.
no-cache	Declare that response should not be cached. This subtly different from an expired response because expiry propagates to dependent resources.
cache-boost	Define a boost factor to increase cache weighting. Must be an integer >0.

DPML

Declarative Process Markup Language (DPML) is a language for orchestration and sequencing or resource requests. DPML defines processing using an XML schema with the following tags:

DPML Tags	
sequence	Either as the root of the document or nested inside another sequence, this tag defines a sequence of processing steps that are evaluated in strict order. A sequence tag may have a **assignment** attribute which assigns it's response to a variable within parent scope. Each step within a sequence may be another sequence, closure, request, literal, comment or one of the built-in endpoints.
closure	Either as the root of the document or nested inside another sequence, this tag defines a set of assignments which are evaluated as necessary to satisfy evaluation of the response assignment. Each assignment may a sequence, closure, request, literal, import, comment or one of the built-in endpoints.
request	Defines a sub-request using declarative request syntax. A request may reference assignments in scope using this:[assignment]. Request arguments may contain identifier strings, literals, or nested sequence, closure or requests. Requests may also issued to closures by specifying an identifier of this:[closure assignment]. The method attributes available on arguments within DPML requests differ slightly from standard requests, there are **lazy** (default), **eager** and **identity**.
literal	Literals can be defined outside the scope of a request.
import	Imports can be specified inside a closure to import assignments from an external resource. The external resource must contain a closure root. Import has an optional **precedence** tag which must have the value of **same** (default), **lower** or **higher**. To determine how imported assignments are merged with existing assignments.
comment	Comment tags can can any content and so can be used to provide code comments or disable blocks of code (because they can be nested unlike XML comments)
declare-tag	Declares a custom tag which will issue a SOURCE request to a specified endpoint. The required **name** attribute specifies the name of the tag to be declared. The text content of the tag should be the meta:[identifier of the endpoint to request]. Arguments within the endpoint grammar are exposed as sub tags within the declared tag.
dereference	Defines a dereference assignment where by the contents of the tag (which may be a request, closure or sequence) provides an identifier which can then be used in subsequent processing.

The DPML module contains a number of built in accessors, these are automatically assigned to tags (in a similar way to declare-tag above):

DPML Built-in endpoints	
if	Perform conditional processing by evaluating the **cond** tag to a boolean. If true the **then** tag is evaluated otherwise the **else** tag is evaluated.
exception	Processes the **try** tag and if any unhandled exceptions are thrown the optional **catch** tag is evaluated. The catch evaluation is provided with the raw exception as **arg:exception,** the exception ID string as **arg:exception-id** and the message as **arg:exception-message**.
throw	Construct and throw an exception with an **id** string, optional **message** string and optional **cause** exception.
log	Log a message. The **message** tag is evaluated to a string and optional **param** tags will be used to replace substitution points marked by %1, %2 etc in the message. Optional **level** tag can be set to INFO (default), WARNING or SEVERE.

modify-response	Specify response expiry and response headers. Used at the end of sequence or closure modify-response takes the existing response as **operand** and a **config** XML document. Example:
	
	`<modify-response assignment="response">`
	`<operand>this:response</operand>`
	`<config>`
	`<literal type="xml">`
	`<config>`
	`<header name="mime">text/plain</header>`
	`<expiry method="CONSTANT">60000</expiry>`
	`</config>`
	`</literal>`
	`</config>`
	`</modify-response>`
	
	Optional expiry method must be one of: ALWAYS, NEVER, DEPENDENT(default), CONSTANT, MIN_CONSTANT_DEPENDENT or MAX_CONSTANT_DEPENDENT.

NetKernel Foundation API

The NKF API is available with endpoints implemented in Java and many of the scripting languages. This section contains the most commonly used interfaces and methods.

INKFRequestContext

INKFRequest	**createRequest**(String identifier) Create a request with a given base identifier.
INKFRequest	**createRequestToEndpoint**(String endpointId) Create a request which is targeted at a specific endpoint.
INKFResponse	**createResponseFrom**(INKFResponseReadOnly resp) Create a response from this endpoint based on this response from a subrequest.
INKFResponse	**createResponseFrom**(Object representation) Create a response from this endpoint based on this representation of the resource.
boolean	**delete**(String identifier) Delete a resource for the given Identifier.
boolean	**exists**(String identifier) Check for the existence of a resource for the given Identifier.
INKFRequest ReadOnly	**getThisRequest**() Return a wrapper around the request which initiated the invocation of this accessor.
INKFAsync RequestHandle	**issueAsyncRequest**(INKFRequest request) Issue an asynchronous request.
Object	**issueRequest**(INKFRequest request) Issue a synchronous request.
INKFResponse ReadOnly	**issueRequestForResponse**(INKFRequest request) Issue the request returning back the whole response so that headers or expiration can be interrogated.
String	**requestNew**(String identifierBase, Object representation) Request the creation of a new resource based on the given identifier and representation.
void	**setNoResponse**() For asynchronous endpoints that don't want to return a response immediately they must call this method to ensure that no synchronous response is returned to the requestee when they complete.
void	**sink**(String identifier, Object representation) Sink a representation to the resource identified by identifier
Object	**source**(String identifier) Source a resource
<T> T	**source**(String identifier, Class<T> representationClass) Source a resource for a specific representation.
INKFResponse ReadOnly	**sourceForResponse**(String identifier) Source the resource returning response.
INKFResponse ReadOnly	**sourceForResponse**(String identifier, Class representationClass) Source the resource returning response.
<T> T	**sourcePrimary**(Class<T> representationClass) Source the primary argument as a specific representation.
INKFResponse ReadOnly	**sourcePrimaryForResponse**(Class representationClass) Source the primary argument return response.
<T> T	**transrept**(Object primary, Class representationClass) Transrept an internally created resource representation into alternative form.
INKFResponse ReadOnly	**transreptForResponse**(Object primary, Class representationClass) Transrept an internally created resource representation into alternative form.

INKFRequest

void	**addArgument**(String name, String identifier) Add a named argument to the request identifier
void	**addArgumentByValue**(String name, Object representation) Add the argument to a pass-by-value space which is injected into the request context.
void	**addArgumentFromResponse**(String name, INKFResponseReadOnly response) Add a named pass-by-value argument to the request with the value and metadata from a previous sub-request or internally constructed.
void	**addPrimaryArgument**(Object representation) Set the primary argument on the request.
void	**addPrimaryArgumentFromResponse**(INKFResponseReadOnly resp) Set the primary argument on the request from a previous sub-request or internally constructed.

void	**injectRequestScope**(ISpace space) Inject an additional space into the scope of the request to execute.
void	**setHeader**(String name, Object representation) Set a header on the request.
void	**setPriority**(int priority) Set the priority header on the request. (constants for this value defined as fields on INKFRequestReadOnly)
void	**setRepresentationClass**(Class representationClass) Indicate the required type (class or interface) of the object returned as the resource representation when this request is processed.
void	**setVerb**(int verb) Sets the verb of the request encoded as an integer value. (constants for this value defined as fields on INKFRequestReadOnly)

INKFRequestReadOnly

boolean	**argumentExists**(String name) Return true if the given argument is on the request.
int	**getArgumentCount**() Return the number of arguments on the request.
String	**getArgumentName**(int index) Get the name of the argument at the given index.
String	**getArgumentValue**(int index) Get the value of the argument at the given index.
String	**getArgumentValue**(String name) Return the value of the first instance of given argument, null if it doesn't exist.
Set<String>	**getHeaderKeys**() Return the set of all header names on this request
Object	**getHeaderValue**(String name) Return the value for a specific header
List<Object>	**getHeaderValues**(String name) Return all the values for a given header.
String	**getIdentifier**() Return the Identifier of the request
Object	**getPrimary**() Returns the primary representation on the request or null if non supplied
INKFResponse ReadOnly	**getPrimaryAsResponse**() Returns the primary response on the request or null if non is supplied
Class	**getRepresentationClass**() Return the representation class that the client expects in the response
int	**getVerb**() Return the request verb. (constants for this value defined as fields on this interface)

INKFResponse

void	**setExpiry**(int type) Set the mode of expiry for this response. (constants for this value defined as EXPIRY_ fields on this interface)
void	**setExpiry**(int type, long timeConstant) Set a time based expiry.
void	**setExpiry**(int type, INKFExpiryFunction function) Set a custom expiry function.
void	**setHeader**(String name, Object representation) Sets a response header.
void	**setMimeType**(String type) Sets the mimetype header.
void	**setNoCache**() Set the NO_CACHE header.

INKFResponseReadOnly

Object	**getHeader**(String name) Return header value.
<T> T	**getHeaderAs**(String name, Class<T> representation) Return header with a specified representation class. Will transrept if necessary.
Iterator<String>	**getHeaderNames**() Return an iterator over header names.
String	**getMimeType**() Return the mime header.
Object	**getRepresentation**() Return the representation in the response
INKFRequest	**getRequest**() Return the request that was issued to give this response
boolean	**hasHeader**(String name) Return true if a header with the given name is present on the response
boolean	**isExpired**() Return true if the response has expired.

INKFAsyncRequestHandle

Object	**join**() Wait indefinitely for an asynchronous subrequest to complete and return representation.
Object	**join**(long timeout) Wait for an asynchronous subrequest to complete and return representation.
INKFResponse ReadOnly	**joinForResponse**() Wait indefinitely for an asynchronous subrequest to complete and return response.
INKFResponse ReadOnly	**joinForResponse**(long timeout) Wait for an asynchronous subrequest to complete and return response.
void	**setListener**(INKFAsyncRequestListener listener) Register a listener to be notified when the response is available.

INKFAsyncRequestListener

void	**receiveException**(NKFException exception, INKFRequest request, INKFRequestContext context) Called when a request fails.
void	**receiveResponse**(INKFResponseReadOnly response, INKFRequestContext context) Called when the response from a request is available.

Developer Tools

The Netkernel Admin Panel http://localhost:1060/ provides a number of useful developer tools:

Key Developer Tools	
Explorer	Browse around deployed modules exploring their sub-structure of spaces and endpoints.
Visualizer	A time machine debugger which captures every request and response so that you can see exactly what is happening during the processing of a request.
Request Trace	Injects requests into a space so that you can test resolution and other behaviours.
Grammar Kitchen	Develop grammars and test their operation parsing, matching and generating identifiers.
Xunit Testing	Develop and execute unit tests for your modules.
Endpoint Profiler	Get detailed information on endpoint execution counts, average and total execution times which aids performance tuning.
New Module Wizard	Takes the effort out of creating a new working module and deploying it.
Cache Viewer	View contents of both representation and resolution caches.

NetKernel Enterprise Edition

NetKernel Enterprise Edition adds assurance to the rock solid NetKernel foundation. It includes professional development, production and management features that take your use of NetKernel to the next level. Enterprise features include:

- Apposite Repository Manager
- NetKernel Protocol - a high performance cloud spanning protocol for bridging ROC instances
- NKP Load balancer
- Virtual Hosting support
- Level 2 Cache backed by database and cluster shareable.
- Highly scalable asynchronous HTTP client / server
- Realtime profiling and reporting
- Deadlock detector and resolver
- Encrypted modules for secure deliver of functionality
- Advanced Visualizer Plugins
- Role based administration panel security

1060 Research recommends NetKernel Enterprise Edition in production environments.

Getting more Information

NetKernel ships with rich library of documentation which covers all the aspects in the reference in more detail as well as many other topics.

Before commencing serious development with NetKernel we would encourage taking out a support subscription which will provide responsive and detailed help with any aspect of NetKernel. For more details see: http://www.1060research.com/services/support/. 1060 Research provides a dedicated customer service portal to efficiently support Enterprise Edition customers.

Questions can also be asked on the NetKernel Discussion Forum at http://www.netkernel.org/forum/ which is read and monitored by the NetKernel team as well as community members.