oTN-2010-01

openlab Summer Student Report

# Shared library implementation for the BDII Information System

Author: Cornel Nicolae Micu
Supervisor: Oliver Keeble

13th August 2010
Version 1
Distribution: **Public**

# Table of contents

## 1 Abstract

The IT-GT-DMS section maintains two software components: Grid File Access Library (GFAL) and File Transfer Service (FTS), which interact in similar ways with the *information system*. Each one had a separate implementation of the relevant logic. The new created component `is-interface` is a refactored specification for a shared library implementation written in C. The aim is to design and implement a Service Discovery API as a common interface which can be adopted by the tools already in the gLite distribution. In the medium term this will lower maintenance costs on the software and improve its stability.

## 2 Introduction[1]

The **Grid information system** is an important component in the grid infrastructure. This provides detailed information about *grid services* needed for various different tasks. The *EGEE information system* has a hierarchical structure of three levels based on the Berkley Database Information Index (BDII)--it can be visualized as an LDAP database. The *resource level* BDII is usually co-located with the *grid service* and provides information about that service. Each grid site[2] runs a *site level* BDII. This aggregates the information from all the *resource level* BDIIs running at that site. The *top level* BDII aggregates all the information from all the *site level* BDIIs and hence contains information about all *grid services*. There are multiple instances of the *top level* BDII in order to provide a fault tolerant, load balanced service. The information system clients query a *top level* BDII to find the information that they require.

[1] For a general description of the CERN information system, please see the TWiki documentation, on which the section 2, 3 & 4 from this report is based on: https://twiki.cern.ch/twiki/bin/view/EGEE/InformationSystem

[2] **Site**, in our context, is *the organization running the grid services*.

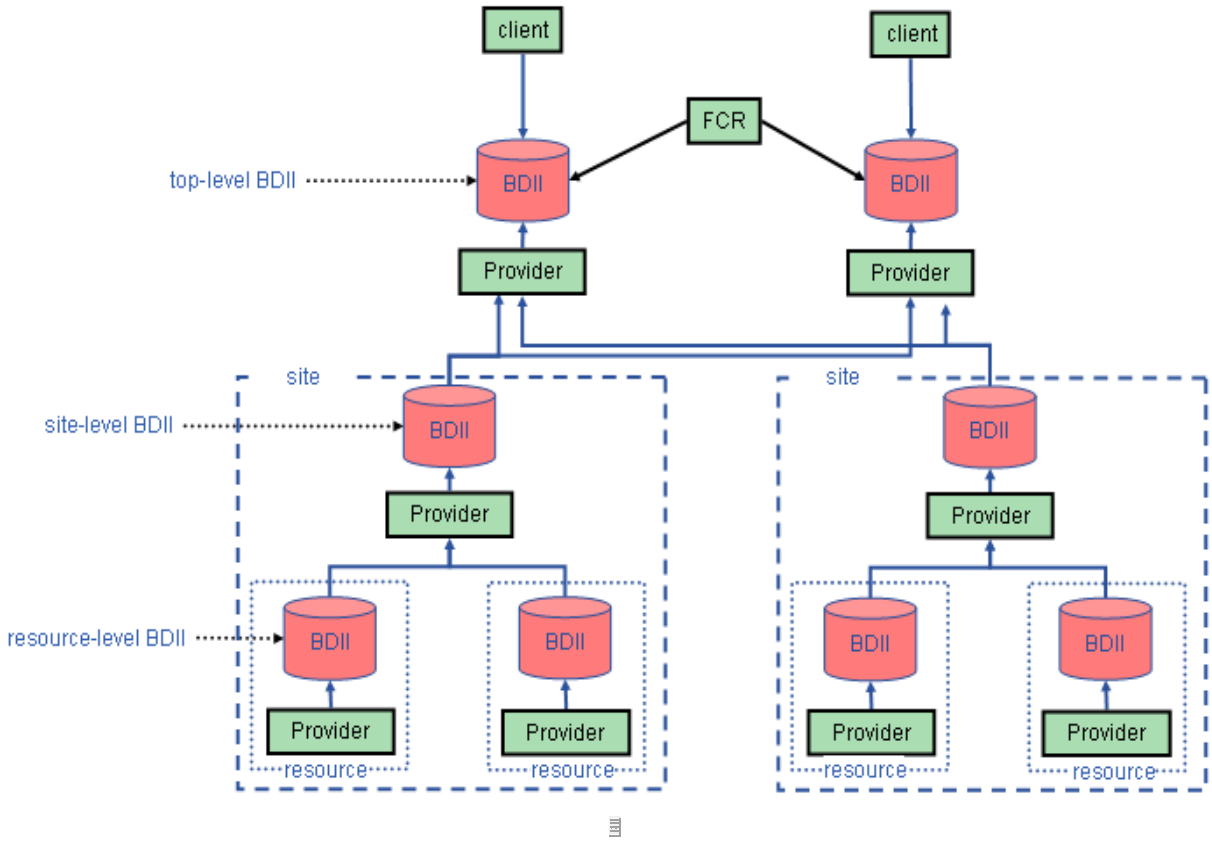


Diagram 1: the BDII hierarchical structure[3]

## 3 BDII—Berkeley Database Information Index

The BDIIs are populated with information by running *information providers*. These are scripts which obtain information, format it as LDAP Data Interchange Format (LDIF) and print the result to standard out.

The information in the information systems conforms to a schema called the **GLUE schema**. A full description of the schema can be found in the specification document and the use of the schema within EGEE is documented here.

The Berkeley Database Information Index (BDII) consists of a standard LDAP database which is updated by an external process. The update process obtains LDIF from a number of sources and merges them. It then compares this to the contents of the database and creates an LDIF file of the differences. This is then used to update the database.

[3] FCR = Freedom of Choice for Resources. All the abbreviations, if not explicit stipulated within the text, are also to be found at the end of the document.

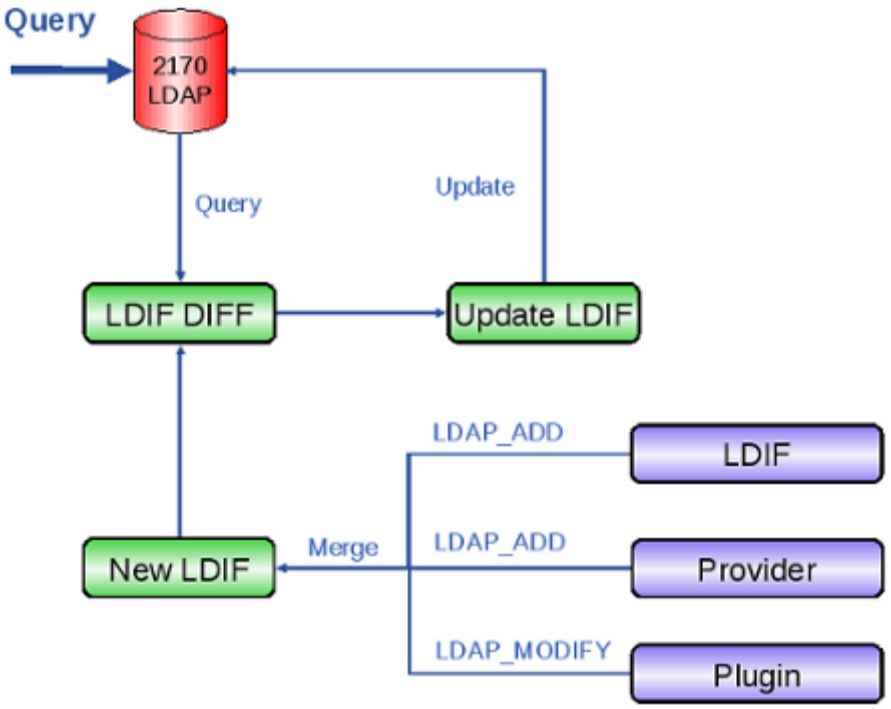


Diagram 2: LDAP Query

## 4 Glue Usage within Enabling Grids for E-sciencE ( EGEE )

A full description of the schema can be found in the specification document and this page describes the usage of the Glue Schema within EGEE. The current version used is version 1.3 and the definitive schema definition can be found here. In addition there is a section in the glite user guide describing the use of the Glue Schema with respect to matchmaking.

| Entity | Inherits from | | Description |
|---|---|---|---|
| Site | | | Set of resources that are installed and managed by the same organization/set of persons (N) |
| **Property** | **Type** | **Mult.** | **Unit** | **Description** |
| UniqueID | string | 1 | | Unique Identifier of the Site (N) |
| Name | string | 1 | | Human-readable name (N) |
| Description | string | 1 | | Short description of this site (N) |
| EmailContact | string | 1 | | The main email contact for the site. Syntax rule: "mailto:" followed by a list of email addresses separated by a comma (e.g.: mailto: email1, email2, email3) (N) |
| UserSupportContact | string | 1 | | E-mail addresses of the support service. Syntax rule: "mailto:" followed by a list of email addresses separated by a comma (e.g.: mailto: email1, email2, email3 (N) |
| SysAdminContact | string | 1 | | E-mail addresses of the system administrator. Syntax rule: "mailto:" followed by a list of email addresses separated by a comma (e.g.: mailto: email1, email2, email3 (N) |
| SecurityContact | string | 1 | | E-mail addresses of the security manager. Syntax rule: "mailto:" followed by a list of email addresses separated by a comma (e.g.: mailto: email1, email2, email3 (N) |
| Location | string | 1 | | Geographical location of this site (e.g., city, state, country) (N) |

Diagram 3: Glue Schema 1.3

## 5 Shared Library specification

The project had to merge, refactor and clean the code responsible for accessing the **Information System** both from GFAL and FTS (see the grey parts in question on the diagram 4).

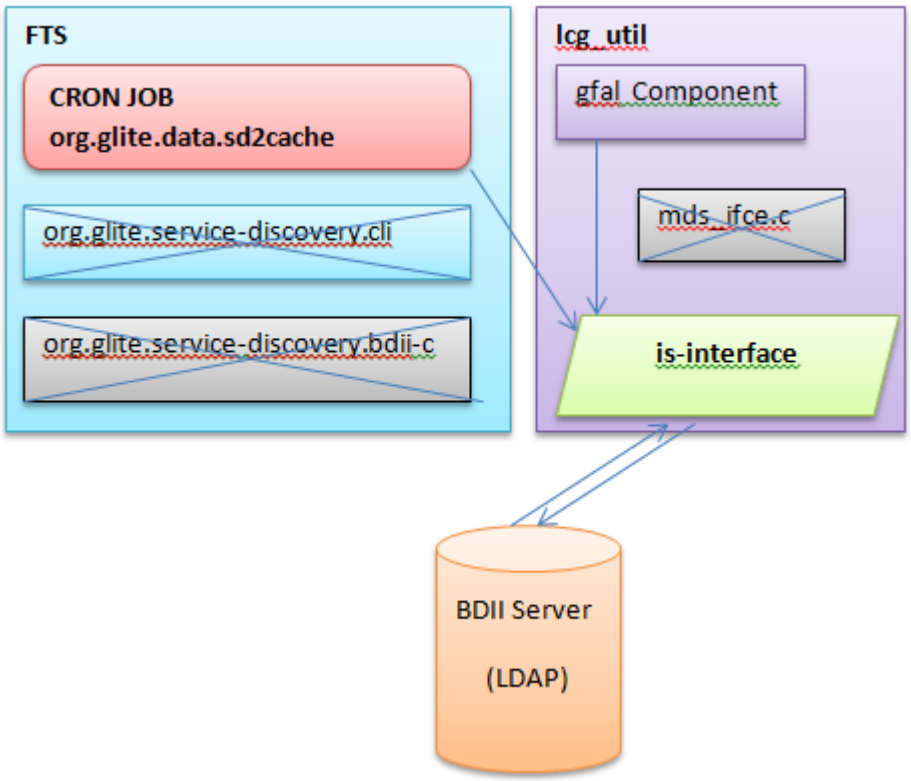


Diagram 4: information system: dependencies and relations

The `is-interface` will be used both in FTS and `gfal`. `Gfal` will be dependent of the new `is-interface` in the same manner as the `org.glite.data.sd2cache`[4] component in FTS will depend. The `org.glite.service-discovery.cli` is run by a `CRON JOB`, where the data retrieved is cached and later used by the different agents. The `gfal` approach is a bit simpler, because the information is received directly from the Information System, where only some `ldap` values of the GLUE Schema are queried (e.g. `lfc_endpoint, storage_path, seap_info, ce_ap, types and endpoints, vo_info, sa_path`). The `is-interface 1.0.0` will contain the functionality of the both `gfal` and FTS interface, sharing the same `ldap` logic.

[4] This component caches the information retrieved from the BDII Server locally. The information flow is as it follows: BDII server is running somewhere on one grid node, the cron job which called this component is deployed on the same node where FTS runs, the job queries all the relevant information, stores it in a local cache (services.xml file), and agents regularly re-read them. The purpose of the job is updating regularly the cache with information from the BDII and reducing the network overhead, because the agents do not need to make LDAP calls every time they need service discovery information.

In order to implement this application programming interface (API), all dependencies of external libraries need to be resolved (`ccheck, ldap, lber, glib`), which are not configured per default.

The CLI, has a `main()` function, which checks the calling parameters, and then call the `tool_doit()` function which then uses the API according the desired parameters. See chapter 9 for a CLI description.

# 6 Project Tasks

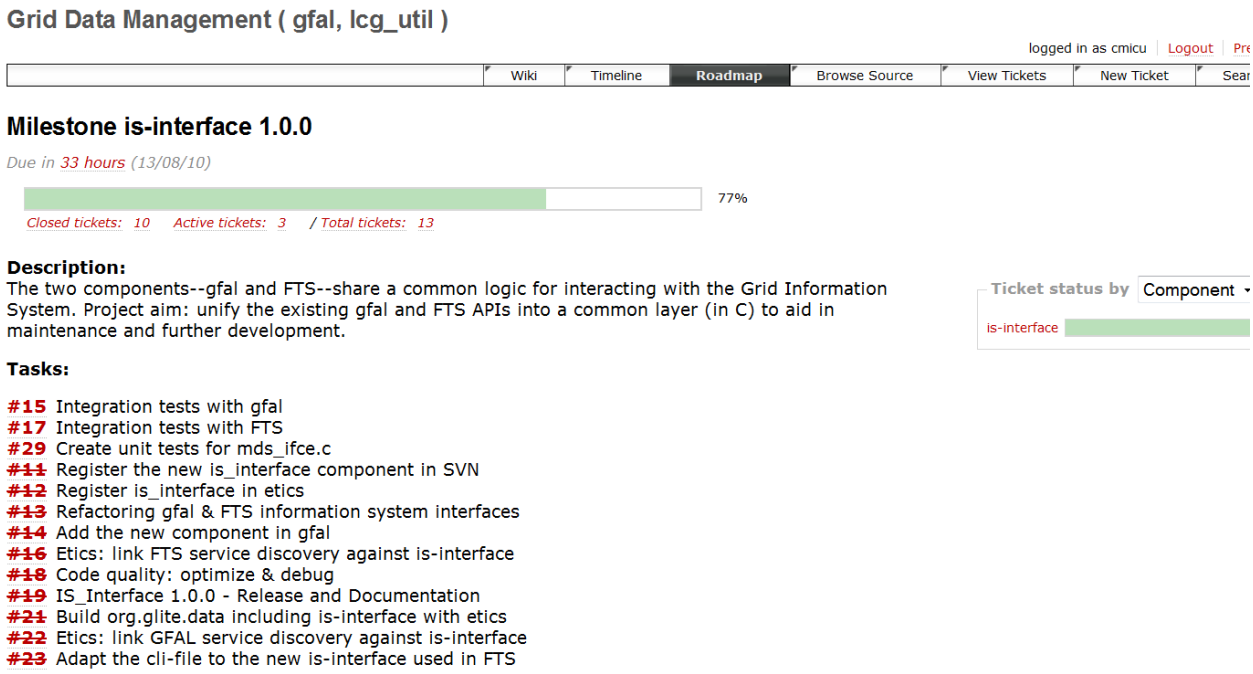


Diagram 5 : is-interface 1.0.0 – TRAC management system

The project was part of a distributed data management system—TRAC (See Diagram 5) and the SVN repository. To the `is-interface 1.0.0` **milestone**, tasks in form of **tickets** were assigned, so that the project progress could be visible to the data management group. For a closer overview of the ticket descriptions, see link: https://svnweb.cern.ch/trac/lcgutil/milestone/is-interface%201.0.0

### Implementation

The is-interface needed two implementations: FTS and gfal. The first one apparently succeeded--to mention that a cron job runs hourly the is-interface and no bug tickets related to this issue by the end of this report were issued. For this implementation, the `org.glite.data.sd2cache,` needed to be changed, so that the is-interface binaries could be found by the sd2cache component.

The gfal implementation requires more settings, this is not implemented yet. There could be two scenarios for this implementation:

a) Check out the `gfal` code and replace/update the function prototypes which are depending of the information system. This can be easily achieved by deleting/commenting the code from mds_ifce.c file and look-up for missing dependencies. Updates of those prototypes according to the new is-interface is needed and also to add the is-interface library as a dependency. Run unittests and integration tests.

b) Remove only the *definition* of the functions from the mds_ifce.c and put the call to the respective function from the `is-interface` instead. This approach is easier but less elegant and can be well implemented as a temporary solution.

**Project Problems and Solutions:**

The gfal code developed a lot lately which raised the level of complexity, so that code maintenance was a difficult task. Besides this, the FTS code responsible for the Service Discovery used the same LDAP logic as the gfal one. Extracting the common layer of these two components required a deep code understanding, and a lot of configuration trials in ETICS, the main build platform. In addition, implementing the new component into the complex system of ETICS requires a global understanding of the dependencies and libraries used, tasks which were not manageable without the (re)learning of things related to LDAP, Eclipse, Unix/Linux Terminal Commands, SVN and of course C. This project is an ongoing one, since probably in the future new features and fixes will be added if required, to a version 2.0.0 of the same interface.

## 7 API Tests

The real endpoint tests require valid credentials (i.e., X.509 certificate).

The new API must have the same outputs as the old FTS and `gfal` APIs, since the functionality provided had to be the same. In the SVN repository in the branches folder, there is the is-interface with unit-tests (they are done locally using ccheck), but not fully implmented. They do not cover the entire API; the ccheck framework is implemented but the injection methods cover only 35-40% from the LDAP functions which need to have an injection method. Besides this, the `gfal` layer needs an extra amount of testing, because for a comparison of the output with the old service discovery output, at least a `unittest` of the later one is required. This part is a future `todo` and will assure a clean implementation of the is-interface in the gfal lcg_util code.

## 8 Command Line Interface ( CLI )

The new `is-interface` API has also a CLI integrated (See in diagram 4 `org.glite.data.service-discovery.cli)`. The CLI is actually an application that executes the BDII queries. It was an external dependency; which is here re-implemented on top of the is-interface. This was the only dependency of FTS on the Infosys. This solution simplified the deployment, maintenance, etc. by obsoleting that dependency.

```
Usage: query [options]
Available options:
      -h          Print this help text and exit.
      -q          Quiet operation.
      -v          Be more verbose.
      -n NAME           Name of the service to query
      -t TYPE           Type of service to query
      -s SITE           Site of service to query
      --host|-H HOST        Host of the service to query
      -e|--print-endpoint         Print endpoints only
      -N|--print-name       Print names only
      -x          Print details, too
      -a          Query associated services
      --xml|-X          Produce xml output to stdout
      -d ATTR=VAL       Query services which match data
```

## 9 Summary

Grid Information System: the Grid File Access Library Interface and File Transfer Service Interface are used to get attributes from the Information System. The Information System is structured respecting the GLUE Schema (GLUE 1.3).
This Schema describes the resources which are able to be discovered for management. The LDAP Library is used by the `is-interface` in order to obtain the information from the BDII Servers. `Is-interface` provides a set of functions (API) plus a CLI, to query the BDII Servers, both from `gfal` and `FTS`.

## 10 References

https://twiki.cern.ch/twiki/bin/view/EGEE/InformationSystem

## 11 Terminology glossary:

* gfal = *Grid* File Access Library
* FTS = File Transfer Service
* BDII = *Berkeley Database Information Index*
* *LDAP = Lightweight Directory Access Protocol*
* *gLite* = middleware stack for grid computing
* SRM = Storage Resource Manager
* GLUE = Grid Laboratory Uniform Environment
* IS = Information System
* SE = Storage Element

## 12 Appendix -- API Description of the Function Prototypes

```
/* GLOBAL DEFINES ************************************************************/

#define SDStatus_SUCCESS 0
#define SDStatus_FAILURE 1

/* Name of the environment variable holding the default VO */
#define SDEnv_VO     "GLITE_SD_VO"
```

```
/* Name of the environment variable holding the default site */
#define SDEnv_SITE   "GLITE_SD_SITE"

/* GLOBAL TYPE DEFINITIONS  ************************************************/

/**
 * The SDServiceData structure holds a single service data item as a
 * keyword value pair.
 */

typedef struct
{
      /** Keyword (not empty and not NULL). */
      char *key;
      /** Value (not NULL but may be empty). */
      char *value;
} SDServiceData;

/**
 * The SDServiceDataList structure holds an array of SDServiceData items
 * and a count.
 */

typedef struct
{
      /** The plugin that allocated the structure. The application should not
touch it. */
      void * const _owner;
      /** The number of data items (may be zero). */
      int numItems;
      /** Array of data items (NULL if and only if numItems is zero). */
      SDServiceData *items;
} SDServiceDataList;

/**
 * The SDVOList structure holds an array of VO names and a count.
 */

typedef struct
{
      /** The number of VO names (may be zero). */
      int numNames;
      /** Array of VO names (NULL if and only if numVOs is zero). */
      char **names;
} SDVOList;

/**
 * The SDService structure holds the basic details about a GLite service. More
 * details can be obtained from a SDServiceDetails structure (see below).
 */

typedef struct
{
      /** The plugin that allocated the structure. The application should not
touch it. */
      void * const _owner;
      /** Unique service name. */
      char *name;
      /** The type of service. */
      char *type;
      /** The endpoint used to contact the service. */
```

```
	char *endpoint;
	/** The service version. */
	char *version;
} SDService;

/**
 * The SDServiceList structure holds a list of Services and a count.
 */

typedef struct
{
	/** The plugin that allocated the structure. The application should not
touch it. */
	void * const _owner;
	/** The number of services (may be zero). */
	int numServices;
	/** Array of services (NULL if and only if numServices is zero). */
	SDService **services;
} SDServiceList;

/**
 * The SDServiceDetails structure holds full details about a GLite service,
 * including those returned in the SDService structure.
 */

typedef struct
{
	/** The plugin that allocated the structure. The application should not
touch it. */
	void * const _owner;
	/** See description in SDService. */
	char *name;
	/** See description in SDService. */
	char *type;
	/** See description in SDService. */
	char *endpoint;
	/** See description in SDService. */
	char *version;
	/** The name of the site that hosts the service. */
	char *site;
	/** The URL of the WSDL for the service (NULL if it is not
	 a Web Service). */
	char *wsdl;
	/** An administration contact e-mail address. */
	char *administration;
	/** The list of VOs supported by this service. */
	SDVOList *vos;
	/** A list of associated services. */
	SDServiceList *associatedServices;
	/** A list of service data (keyword/value pairs). */
	SDServiceDataList *data;

//GFAL

} SDServiceDetails;

/**
 * The SDServiceDetailsList structure holds a list of Service details and a
count.
 */
```

```
typedef struct
{
      /** The plugin that allocated the structure. The application should not
touch it. */
      void * const _owner;
      /** The number of service details (may be zero). */
      int numServiceDetails;
      /** Array of service datils (NULL if and only if numServiceDatils is
zero). */
      SDServiceDetails **servicedetails;
} SDServiceDetailsList;

/**
 * The SDException structure holds the status of a call to this Service
 * Discovery API.
 */

typedef struct
{
      /** API call status. Will be SDStatus_SUCCESS (guaranteed to be zero)
       on success, or some other value on error. */
      int status;
      /** Reason for failure. Will be NULL if and only if status is
       SDStatus_SUCCESS, but may be an empty string. */
      char *reason;
} SDException;

/*****************************************************************
 * Prototypes
 */

int get_lfc_endpointtt();
int NS_get_storage_path();

/* PROTOTYPE FUNCTIONS
*****************************************************/

/**
 * The sd_bdii_getService function returns basic details about the requested
 * service.
 *
 * You can dispose of any data returned by calling SD_freeService.
 * You can dispose of exception data (on failure) by calling
 * sd_bdii_freeException.
 *
 * @param  serviceName        Unique name of service.
 * @param  exception          If not NULL, receives status of API call.
 *
 * @return Basic service details, or NULL if service cannot be found or if the
 *         API call fails.
 */
SDService *sd_bdii_getService(const char *serviceName, SDException *exception);

/**
 * The SD_getServiceDetails function returns full details about the requested
 * service.
 *
 * You can dispose of any data returned by calling sd_bdii_freeServiceDetails.
 * You can dispose of exception data (on failure) by calling
sd_bdii_freeException.
 *
```

```
 * @param  serviceName        Unique name of service.
 * @param  exception          If not NULL, receives status of API call.
 *
 * @return Full service details, or NULL if service cannot be found or if the
 *         API call fails.
 */

SDServiceDetails *sd_bdii_getServiceDetails(const char *serviceName,
            SDException *exception);

/**
 * The sd_bdii_getServiceData function returns all service keyword/value data
for
 * the requested service.
 *
 * You can dispose of any data returned by calling sd_bdii_freeServiceData.
 * You can dispose of exception data (on failure) by calling
sd_bdii_freeException.
 *
 * @param  serviceName        Unique name of service.
 * @param  exception          If not NULL, receives status of API call.
 *
 * @return Service data list or NULL if the API call fails. Service data list
 *         will be empty if the service cannot be found, or doesn't have any
 *         keyword/value data.
 */

SDServiceDataList *sd_bdii_getServiceData(const char *serviceName,
            SDException *exception);

/**
 * The sd_bdii_getServiceDataItem function returns the value of the requested
 * service parameter.
 *
 * You can dispose of any data returned by calling free().
 * You can dispose of exception data (on failure) by calling
sd_bdii_freeException.
 *
 * @param  serviceName        Unique name of service.
 * @param  key                Parameter name.
 * @param  exception          If not NULL, receives status of API call.
 *
 * @return Parameter value or NULL if the service cannot be found, doesn't
 *         contain the requested data, or if the API call fails.
 */

char *sd_bdii_getServiceDataItem(const char *serviceName, const char *key,
            SDException *exception);

/**
 * The sd_bdii_getServiceSite function returns the name of the site where a
service
 * runs.
 *
 * You can dispose of any data returned by calling free().
 * You can dispose of exception data (on failure) by calling
sd_bdii_freeException.
 *
 * @param  serviceName        Unique name of service.
 * @param  exception          If not NULL, receives status of API call.
 *
```

```
 * @return Site name or NULL if the service cannot be found, or if the API call
 *         fails.
 */

char *sd_bdii_getServiceSite(const char *serviceName, SDException *exception);

/**
 * The sd_bdii_getServiceWSDL function returns the URL to the service WSDL (if
any).
 *
 * You can dispose of any data returned by calling free().
 * You can dispose of exception data (on failure) by calling
sd_bdii_freeException.
 *
 * @param  serviceName        Unique name of service.
 * @param  exception          If not NULL, receives status of API call.
 *
 * @return WSDL string or NULL if the service cannot be found, is not a Web
 *         Service, or if the API call fails.
 */

char *sd_bdii_getServiceWSDL(const char *serviceName, SDException *exception);

/**
 * The sd_bdii_listAssociatedServices function returns a list of services that
are
 * are associated with the requested service and that match the specified type,
 * site and VOs (services with no VO affiliation match any VO specified by the
 * user).
 *
 * You can dispose of any data returned by calling sd_bdii_freeServiceList.
 * You can dispose of exception data (on failure) by calling
sd_bdii_freeException.
 *
 * @param  serviceName        Name of service with which others are associated.
 * @param  type               Type of services required (NULL => any).
 * @param  site               Site of services required (NULL => any).
 * @param  vos                List of VOs from which services may come
 *                            (NULL => any).
 * @param  exception          If not NULL, receives status of API call.
 *
 * @return List of matching services or NULL if the API call fails. List will
 *         be empty if no services are found.
 */

SDServiceList *sd_bdii_listAssociatedServices(const char *serviceName,
            const char *type, const char *site, const SDVOList *vos,
            SDException *exception);

/**
 * The sd_bdii_listServices function returns a list of services that match
 * the specified type, site and VOs (services with no VO affiliation match
 * any VO specified by the user).
 *
 * You can dispose of any data returned by calling sd_bdii_freeServiceList.
 * You can dispose of exception data (on failure) by calling
sd_bdii_freeException.
 *
 * @param  type               Type of services required (NULL => any).
 * @param  site               Site of services required (NULL => any).
 * @param  vos                List of VOs from which services may come
```

```
 *                              (NULL => any).
 * @param  exception            If not NULL, receives status of API call.
 *
 * @return List of matching services or NULL if the API call fails. List will
 *         be empty if no services are found.
 */

SDServiceList *SD_listServices(const char *type, const char *site,
            const SDVOList *vos, SDException *exception);

/**
 * The sd_bdii_listServicesByData function returns a list of services that match
 * the specified keyword/value data, type, site and VOs (services with no VO
 * affiliation match any VO specified by the user).
 *
 * You can dispose of any data returned by calling sd_bdii_freeServiceList.
 * You can dispose of exception data (on failure) by calling
sd_bdii_freeException.
 *
 * @param  data                 List of keyword/value data to match.
 * @param  type                 Type of services required (NULL => any).
 * @param  site                 Site of services required (NULL => any).
 * @param  vos                  List of VOs from which services may come
 *                              (NULL => any).
 * @param  exception            If not NULL, receives status of API call.
 *
 * @return List of matching services or NULL if the API call fails. List will
 *         be empty if no services are found.
 */

SDServiceList *sd_bdii_listServicesByData(const SDServiceDataList *data,
            const char *type, const char *site, const SDVOList *vos,
            SDException *exception);

/**
 * The sd_bdii_listServicesByHost function returns a list of services that
 * match the specified type, the given host and port and VOs (services
 * with no VO affiliation match any VO specified by the user).
 *
 * The host:port is looked up in the service endpoint and a string match is
 * applied in the query (this is more efficient than performing this search
 * on the client side)
 *
 * You can dispose of any data returned by calling sd_bdii_freeServiceList.
 * You can dispose of exception data (on failure) by calling
sd_bdii_freeException.
 *
 * @param  type                 Type of services required (NULL => any).
 * @param  host                 Host name of services required (NULL => any).
 * @param  vos                  List of VOs from which services may come
 *                              (NULL => any).
 * @param  exception            If not NULL, receives status of API call.
 *
 * @return List of matching services or NULL if the API call fails. List will
 *         be empty if no services are found.
 */

SDServiceList *sd_bdii_listServicesByHost(const char *type, const char *host,
            const SDVOList *vos, SDException *exception);

/**
```

```
 * Frees all memory associated with an SDServiceDataList structure.
 *
 * @param  serviceDataList    Structure to free (if NULL, does nothing).
 *
 * @return Nothing.
 */

void sd_bdii_freeServiceDataList(SDServiceDataList *serviceDataList);

/**
 * Frees all memory associated with an SDService structure.
 *
 * @param  service            Structure to free (if NULL, does nothing).
 *
 * @return Nothing.
 */

void sd_bdii_freeService(SDService *service);

/**
 * Frees all memory associated with an SDServiceList structure.
 *
 * @param  serviceList        Structure to free (if NULL, does nothing).
 *
 * @return Nothing.
 */

void sd_bdii_freeServiceList(SDServiceList *serviceList);

/**
 * Frees all memory associated with an SDServiceDetails structure.
 *
 * @param  serviceDetails     Structure to free (if NULL, does nothing).
 *
 * @return Nothing.
 */

void sd_bdii_freeServiceDetails(SDServiceDetails *serviceDetails);

/**
 * Frees all memory associated with SDException structure.
 *
 * @param  exception          Structure to free (if NULL, does nothing).
 *
 * @return Nothing.
 */

/* MEMORY MANAGEMENT FUNCTIONS  ********************************************/

void SD_freeException(SDException *exception);
void sd_bdii_freeService(SDService *service);
void sd_bdii_freeServiceList(SDServiceList *list);
void sd_bdii_freeServiceDataList(SDServiceDataList *list);
void sd_bdii_freeServiceDetails(SDServiceDetails *details);
void sd_bdii_freeVOList(SDVOList *vos);
void sd_bdii_freeServiceDetailsList(SDServiceDetailsList *servicedetailsList);
```